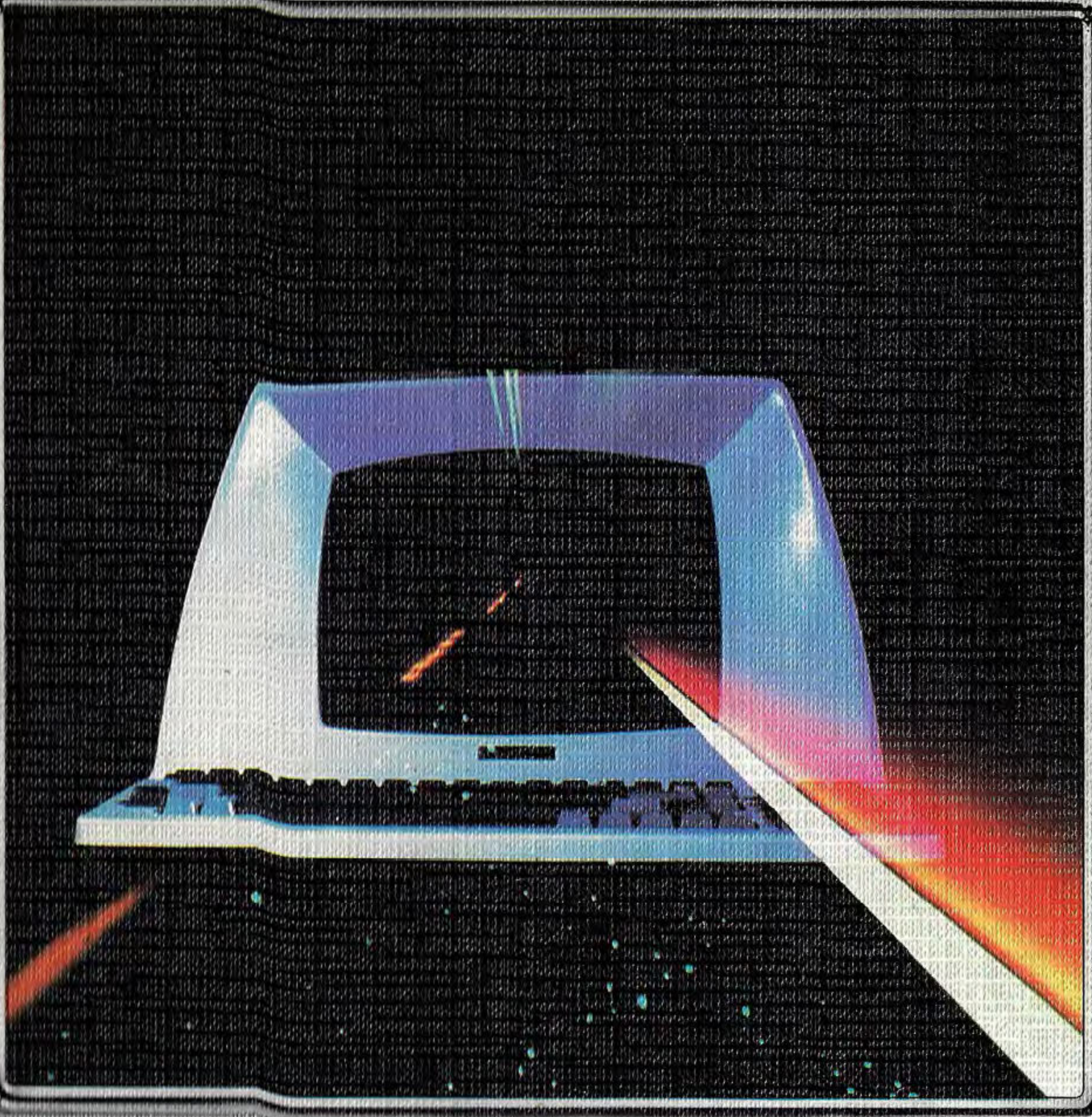
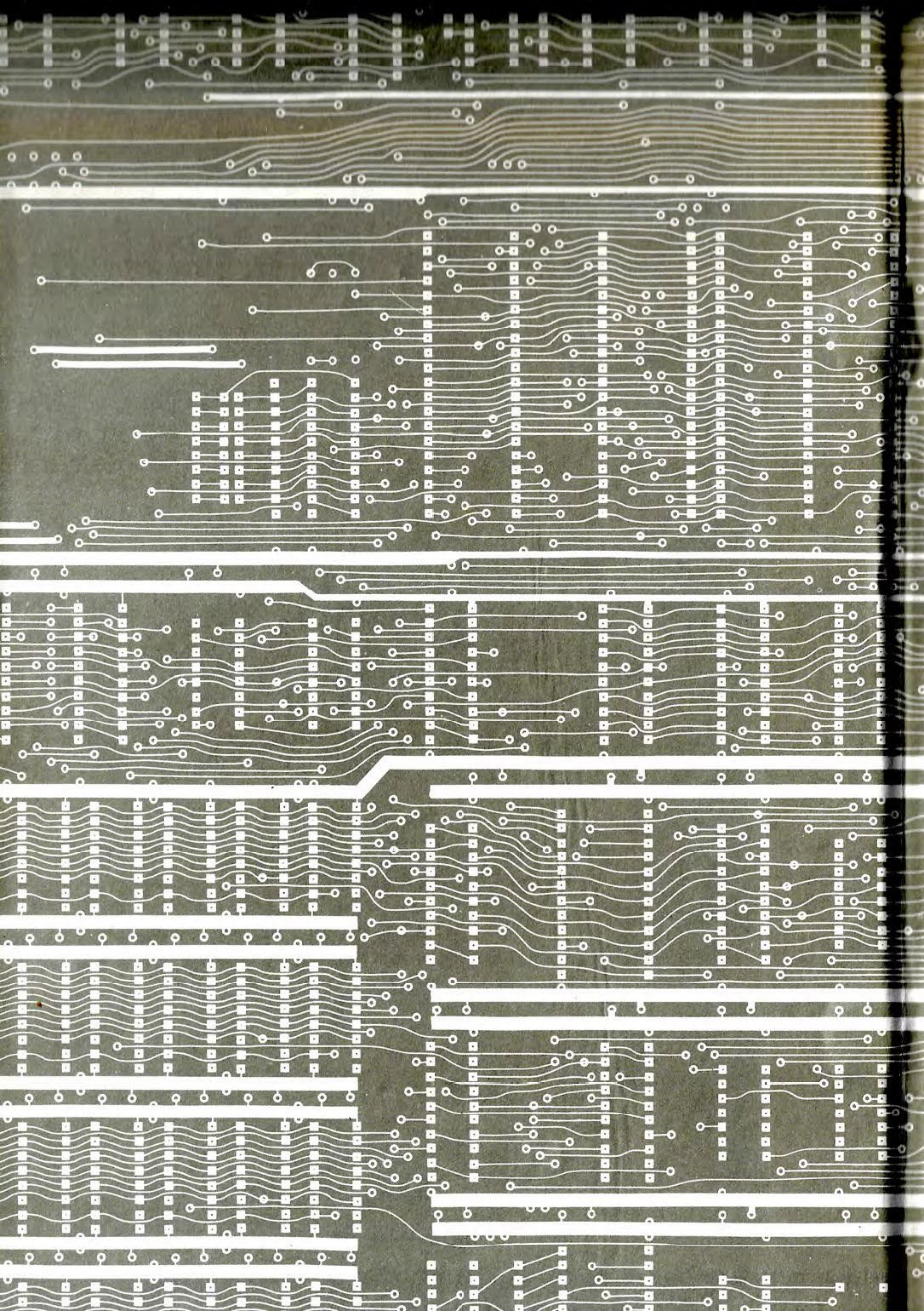
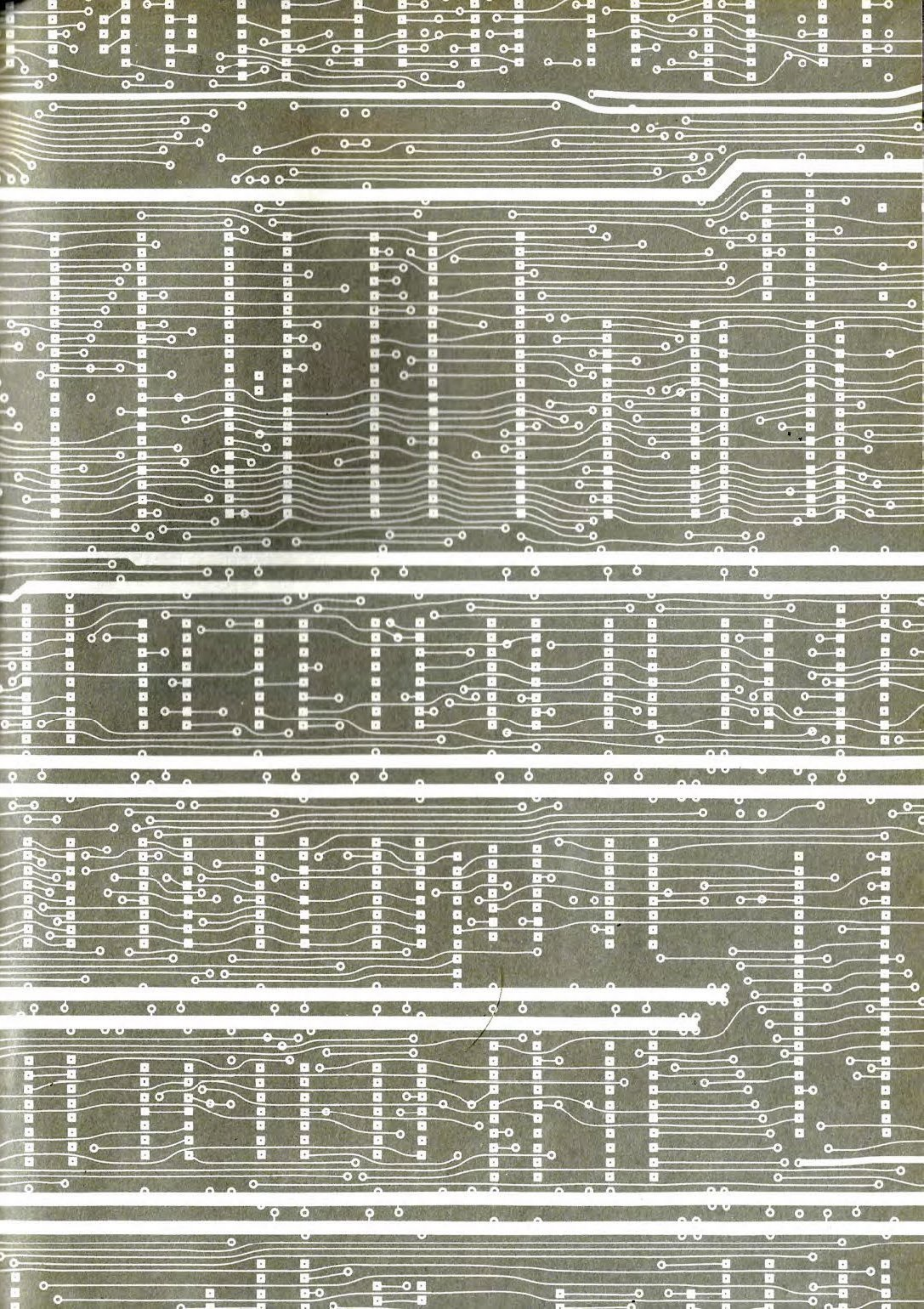
BASIC

ENCICLOPEDIA DE LA INFORMATICA
MINIORDENADORES Y ORDENADORES PERSONALES



EDICIONES FORUM

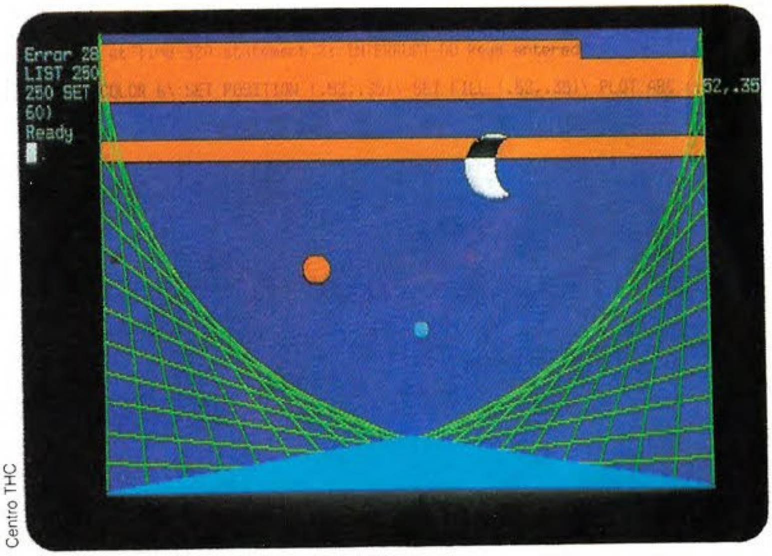




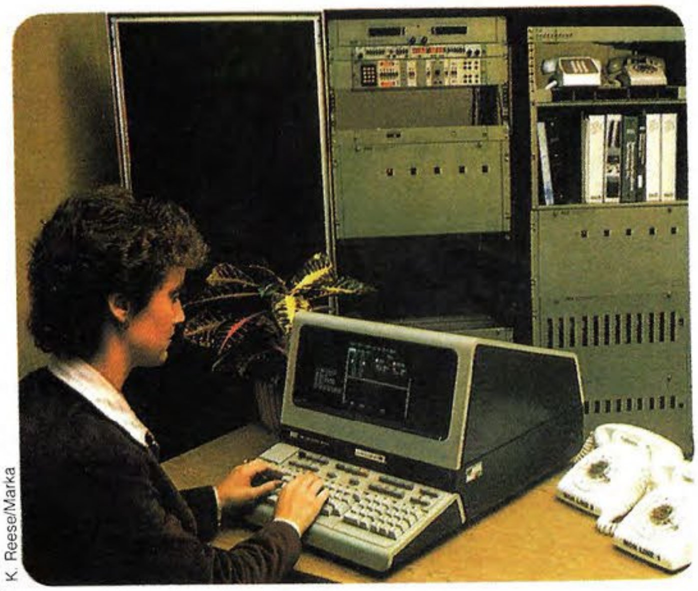
BASIC

ENCICLOPEDIA DE LA INFORMATICA.
MINIORDENADORES Y ORDENADORES PERSONALES

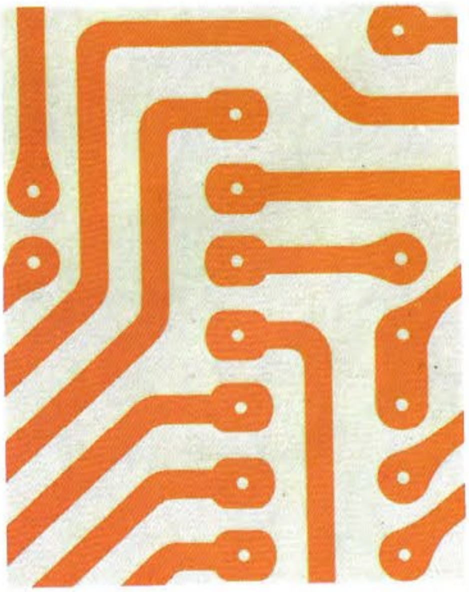
ENCICLOPEDIA DE LA INFORMATICA. MINIORDENADORES Y ORDENADORES PERSONALES



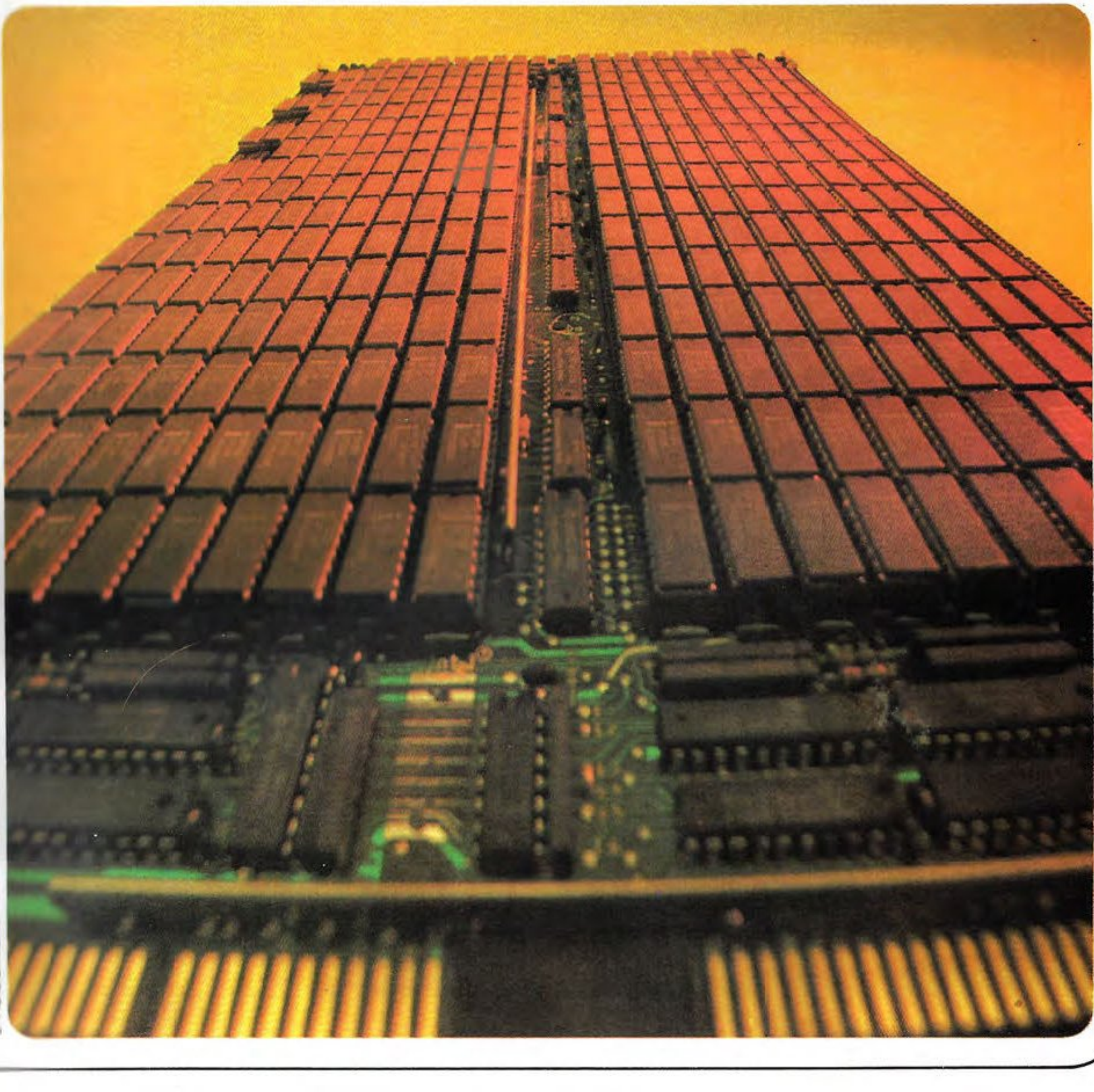
Centro THC



K. Reese/Marka



BASIC 5



J. Pickerell/Marka

BASIC

ENCICLOPEDIA DE LA INFORMATICA. MINIORDENADORES Y ORDENADORES PERSONALES

Presidente
José Manuel Lara

Director Ejecutivo
Jesús Domingo

Dirección editorial
R.B.A., Proyectos Editoriales, S.A.

Dirección técnica
Sante Senni

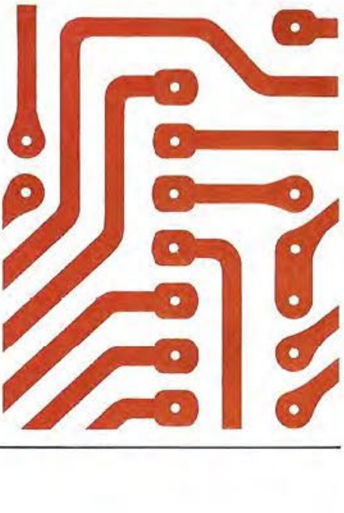
Con el asesoramiento de la Sociedad **E.G.S.**

REDACCION

Dirección editorial: Gabriella Costarelli
Redactor jefe: Marcella Marcaccini
Secretaria de redacción: Giulia Abriani, Giovanna Aloisi
Revisión: Ugo Spezia
Corrección: Maria Albergo, Laura Salvini, Graziella Tassi
Recopilación material gráfico: Carla Bertini, Rossella Pozza
Producción: Piergiorgio Palma
Secretaria de producción: Maria Rita Ciucci
Diseño y dirección artística: Vittorio Antinori
Jefe estudio gráfico: Roberto Sed
Compaginación: Alberto Berni, Riccardo Catani, Patrizia Fazio
Dibujos: Renato Lazzarini, Gianni Mazzoleni, Rolando Mazzoni
Traducción: Carlo Frabetti
Diseño cubierta: Neslé Soulé
Jefe de Producción: Ricardo Prats

Composición electrónica: ITC-Fototipo. Barcelona-Madrid.
Imprime: CAYFOSA - Carretera de Caldas, Km. 3,7 - Santa Perpetua de Mogoda (Barcelona)
Depósito Legal: B. 37.099/83
ISBN (Obra completa): 84-7574-040-5
ISBN (Volumen V): 84-7574-278-5
ISBN (Fascículos): 84-7574-044-8
Impreso en Espa;a - Printed in Spain
Distribuye: Marco Ibérica, Distribuidora de Ediciones, S.A.
Carretera de Irún, Km. 13,350, variante de Fuencarral, MADRID-34

El editor agradece la colaboración de:
Alfa Romeo, Buffetti Data, Commodore Italiana, CPT Italia, Creazioni Walt Disney, Data General, Digital, Doxa, Elsag, Ericsson, Facit Data Products, Ferrari, FIAT, Harden Italia, Hengstler Italia, Hewlett Packard, IBM, Intema, IRET Informática, Italcable, Lancia, Litton BEI, MEE, MSI Data Italia, Olivetti, Perkin-Elmer, Plessey Trading, Prime Italia, Rank Xerox, Rhône-Poulenc Italia, Sanco Ibex Italia, Sarin, Selca Elettronica, Selenia, Sperry, SIP, Telespazio.



El lenguaje Fortran

El Fortran (contracción de FORMULA TRANSLATOR) ha sido uno de los primeros lenguajes de alto nivel orientados al uso científico. La primera versión oficial aparece en 1957, y a ella se han aportado numerosas modificaciones y nuevas implantaciones que han conducido al moderno Fortran ANSI 77, bajo ciertos aspectos muy similar al Basic. La sigla ANSI proviene de las iniciales de American National Standard Institute que, en 1977, estuvo a cargo de la formulación de las últimas modificaciones. La versión ANSI 77 es un estándar pero no un vínculo; en consecuencia, es fácil encontrar formas de Fortran que se diferencian más o menos de ésta.
En la época en que nacieron el Fortran y el Cobol, la introducción de las instrucciones se producía casi exclusivamente utilizando un soporte físico: la ficha perforada. El ordenador utilizaba como periférico de entrada no el teclado, sino el lector de fichas: cada instrucción debía ser tecleada en el teclado de una perforadora de fichas (máquina que no estaba conectada de ninguna manera al ordenador) respetando un formato preciso para que el lector de fichas, y después el Compilador, pudiesen interpretar unívocamente el significado de la instrucción.
El formato de perforación de las instrucciones en las fichas, que han permanecido en uso hasta hace pocos años y en algunos casos particulares todavía lo están, debe contemplarse debidamente en los Compiladores Fortran.
El advenimiento del soporte magnético ha producido, en el curso del último decenio, la gradual eliminación de los voluminosos archivos de fichas, y la memorización de datos y programas se hace en disco o en cinta. Sin embargo, el formato de las instrucciones reconocido por los Compiladores sigue siendo el mismo. Las diferentes partes (campos) que componen las instrucciones hoy continúan siendo tecleadas en una longitud total de 80 columnas, situadas según el formato válido para las fichas.
Por este motivo, tal como se ha hecho con el Cobol, se hará referencia a cada carácter senci-

Un lector de fichas IBM 3504. Hasta hace pocos años, los lectores de fichas eran los periféricos estándar de entrada.



IBM

llo de una instrucción introducida por teclado indicando la correspondiente posición que éstos ocuparían sobre una ficha si se utilizase este tipo de entrada (columna 1, columna 2, etc.).
El lenguaje Fortran, dada la gran semejanza que presenta con respecto al Basic, se presentará utilizando ampliamente la analogía con este último, evidenciando de vez en cuando las eventuales diferencias.

Características fundamentales en relación al Basic

Una característica fundamental del Fortran es la de ser un lenguaje sólo compilado, y por tanto no posee la facilidad de uso y de depuración propia del Basic interpretado. Una segunda diferencia notable es en la numeración de las líneas: sabemos que en el Basic cada línea debe estar numerada progresivamente; en Fortran, la numeración no es obligatoria, y se usa sólo si el programador la necesita para identificar una línea definida (por ejemplo, para el uso de la instrucción GOTO...).
Como falta la obligación de los números de línea, las subrutinas deben identificarse de manera diferente a la del Basic.
Mientras en el Basic las subrutinas están direccionadas refiriéndose al número de línea inicial (p. e. GOSUB 1000), en Fortran tienen un nombre simbólico elegido por el programador.
Por ejemplo, una subrutina que efectúa la suma A = B + C en Basic puede escribirse de la siguiente manera

```
1000 REM Suma
1010 A = B + C
1020 RETURN
```

En Fortran, la primera línea, donde empieza la subrutina, debe contener la palabra SUBROUTINE seguida del nombre simbólico que se usará para llamar dicha subrutina. Por ejemplo, asignándole el nombre SUMA, el equivalente Fortran del programa examinado se convierte en:

```
SUBROUTINE SUMA
A = B + C
RETURN
```

La subrutina así escrita utiliza las tres variables A, B, C.
En el Basic estándar, cada instrucción del programa (comprendidas las instrucciones que componen las subrutinas) puede acceder a todas las variables. En nuestro caso pueden asignarse valores a las variables B y C en una línea cualquiera del programa, calcular su suma en una subrutina (la 1000) y utilizar el resultado (A) en otra zona del programa.
En Fortran esto no es posible a menos que se utilicen instrucciones específicas: **cada subrutina utiliza localmente las propias variables, que por tanto sólo sirven en su interior.**
La transferencia de los valores de las variables entre las subrutinas puede hacerse:

1 / transfiriendo los valores como parámetros
2 / para declaraciones de COMMON

Para llamar una subrutina en Fortran debe utilizarse la instrucción CALL... en lugar de la GOSUB... del Basic.
Por ejemplo, la instrucción de llamada de la subrutina antes considerada es CALL SUMA (en Basic habría sido GOSUB 1000).
Para transferir las variables como parámetros debe especificarse su nombre simbólico bien en la llamada, bien en la primera instrucción de la subrutina que las utiliza.
En el ejemplo anterior la llamada se convierte en

```
CALL SUMA (A, B, C)
```

y la subrutina (de nombre SUMA) debe empezar con la instrucción

```
SUBROUTINE SUMA (A, B, C)
```

La comparación entre las versiones Basic y Fortran de un mismo programa que utiliza la subrutina indicada se ha representado en la tabla de la página siguiente.
En la versión Fortran se han eliminado todos los números de línea, a excepción del primero (línea 10), que es el punto de llegada del GOTO (línea 40 Basic y la equivalente Fortran). Para ser identificada, la línea de llegada debe tener un nombre, es decir una etiqueta (label).
En Fortran, la eventual numeración de las líneas sólo tiene el significado de etiqueta y no debe seguir ninguna lógica: la línea 10 puede ir antes que la 5 y continuar con la 1000. También en Basic es posible utilizar numeraciones con incremento variable, pero los valores deben estar dispuestos en orden creciente. El segundo método de transferencia de los valores consiste en

VERSION BASIC Y VERSION FORTRAN DE UN MISMO PROGRAMA

Versión Basic

```
10 C = 10
20 B = 4.7
30 GOSUB 1000
40 GOTO 10
1000 REM Suma
1010 A = B + C
1020 RETURN
```

Versión Fortran

```
10 C = 10
   B = 4.7
   CALL SUMA (A,B,C)
   GOTO 10
   SUBROUTINE SUMA (A,B,C)
   A = B + C
   RETURN
```

declarar «comunes» los parámetros a transferir mediante la instrucción COMMON, análoga a la del Basic.
El programa anterior se convierte en:

```
     COMMON A, B, C
  10 C = 10
     B = 4.7
     CALL SUMA
     GOTO 10
     SUBROUTINE SUMA
     COMMON A, B, C,
     A = B + C
     RETURN
```

En realidad, el programa considerado en los ejemplos contiene un error. El Compilador está informado del punto de inicio de la subrutina (SUBROUTINE SUMA) y del punto final de la misma (RETURN), pero no conoce el principio del programa ni su final. En Basic, el inicio del programa corresponde a la línea con numeración más baja. En Fortran debe especificarse el punto de inicio con la instrucción PROGRAM seguida del nombre que se da al programa:

```
1        PROGRAM PRUEBA
2     C  EJEMPLO
3        COMMON A, B, C
4    10  C = 10
5        B = 4.7
6        CALL SUMA
7        GOTO 10
8        STOP
9        END
10       SUBROUTINE SUMA
11       COMMON A, B, C,
12       A = B + C
13       RETURN
14       END
```

La numeración progresiva que aparece a la izquierda de las instrucciones no forma parte del programa y sólo sirve para identificar las líneas para eventuales correcciones. Normalmente, la máquina la suministra automáticamente para facilitar el trabajo del programador.
El programa empieza con la instrucción PROGRAM... (línea 1); la línea 2 contiene un comentario que puede indicarse de diferentes maneras. Los símbolos más usados (como el REM o el símbolo ' del Basic) son C ! * y se teclean como primer carácter de la instrucción (en la columna 1 de una hipotética ficha perforada).
El Basic no tiene un formato físico para introducir las instrucciones: cada línea empieza con un número y sigue con la instrucción, y es posible interponer uno o más espacios entre el número de línea y la instrucción propiamente dicha.
Por ejemplo, éstas son instrucciones válidas:

```
  10 A = B + C
  10    A = B + C
1000    A = B + C
```

En Fortran, las instrucciones deben empezar en la columna 7 (de una hipotética ficha), los comentarios deben tener el símbolo adecuado en la columna 1 y las líneas de continuación (cuando una instrucción no cabe en una sola línea) un carácter cualquiera (excluidos el 0 y el espacio) en la columna 6.
En la tabla de la página siguiente se indican algunas diferencias entre el Fortran y el Basic.

Operadores aritméticos, lógicos y relacionales

En lo que respecta a los operadores aritméticos, el cálculo de las expresiones se desarrolla con las mismas reglas y símbolos del Basic, excepto

ALGUNAS DIFERENCIAS ENTRE FORTRAN Y BASIC

	FORTRAN	BASIC
Números de línea	Opcionales; no deben seguir ningún orden	Necesarios en orden creciente
Principio programa	Necesario e identificado con un nombre con la instrucción PROGRAM nombre	Empieza siempre por la línea con numeración más baja
Subrutinas	Identificadas por un nombre simbólico precedido de la palabra reservada SUBROUTINE. Se terminan con las instrucciones RETURN y END	Identificadas por un número de línea. Se terminan con la instrucción RETURN
Llamada a subrutinas	Instrucción CALL seguida del nombre simbólico: CALL PRUEBA	Instrucción GOSUB seguida del número de línea: GOSUB 1000
Transferencia de valores	Como parámetros: CALL PRUEBA (A,B,C). Como declaración común: COMMON A,B,C	No necesaria en términos explícitos. Cada punto del programa accede a todas las variables

la elevación a potencia que se indica con el símbolo * * (el mismo que el utilizado en Cobol).
Los operadores relacionales son los mismos del Basic, con algunas diferencias por la sintaxis.
Mientras que en el Basic estos operadores se representan simbólicamente (>, <, =), en el Fortran se indican con una sigla compuesta normalmente de dos letras, y deben ir precedidos y seguidos por el símbolo . (punto).
También los operadores lógicos son los mismos que los encontrados en el Basic, pero también para éstos, la diferente sintaxis del Fortran prevé la presencia de los puntos a la derecha y a la izquierda del código mnemónico.
Los operadores lógicos y relacionales se han resumido en la tabla de la página siguiente.

Variables y constantes en el Fortran

El lenguaje Fortran acepta por lo menos los siguientes tipos de datos:

— Enteros
— Reales en simple precisión
— Reales en doble precisión
— Complejos en simple precisión
— Complejos en doble precisión
— Lógicos (booleanos)
— Caracteres

Algunas formas prevén los tipos «entero acortado» (short integer) y «lógico acortado» (short logical), los cuales ocupan un número menor de bits y, por tanto, en ciertas condiciones, permiten ahorrar espacio en memoria.
El significado de cada tipo es análogo al visto en el Basic, a excepción de los números complejos, que no encuentran correspondencia en el Basic estándar, y de los enteros, para los cuales el Fortran prevé una declaración implícita: en Fortran todas las variables que empiezan con I, J, K, L, M, N se suponen enteras por omisión salvo indicación contraria.
Esta diferencia con respecto al Basic tiene una importancia marginal; el programador adquiere rápidamente la costumbre de utilizar para las variables enteras las letras especificadas, y a menudo las emplea también en Basic.
Las variables complejas son una característica peculiar del Fortran (aunque algunas formas del Basic las prevén), consecuencia del uso eminentemente científico de este lenguaje.
El uso de las variables complejas requiere conocimientos que van más allá de la programación, y está limitada a sectores específicos.
Los datos de tipo lógico tienen las mismas funciones que en el Basic; pueden asumir sólo los valores **verdadero** (TRUE) o **falso** (FALSE), normalmente representados respectivamente con los valores 1 y 0 como en Basic. Sin embargo, la

sintaxis es diferente, dado que, en Fortran, las palabras TRUE o FALSE deben ir precedidas y seguidas de un punto: .TRUE. y .FALSE.
Además de las variables y constantes numéricas, el Fortran prevé el uso de las constantes

— Hexadecimales
— Octales
— Hollerith

Las constantes octales y hexadecimales tienen el mismo significado que el visto en Basic, aunque se utilizan con una simbología diferente.
La notación más empleada para las constantes octales es la letra B después del número

10B = 8 decimal

mientras que para las constantes hexadecimales es

Z 'valor'

donde con «valor» se indica la expresión hexadecimal del valor de la constante (Z'A' = 10 decimal).
Las constantes Hollerith son una forma particular de cadena. La sintaxis es

nHxxxx

donde n es el número de caracteres de la cadena (xxxx) y H el símbolo que indica el tipo particular de constante (abreviación de Hollerith).
Por ejemplo,

19HESTA ES UNA PRUEBA

es una forma válida de constante Hollerith. Téngase en cuenta que los espacios insertados entre una palabra y otra deben contarse como caracteres.
El tratamiento de las cadenas en Fortran es algo diferente del Basic. Antes que nada debe decirse que en Fortran no existen variables de tipo cadena, y por ende no se utilizan nombres con el símbolo $. Los caracteres de una cadena se memorizan en variables enteras, y por tanto ocupan 8 bits cada uno. En una variable entera, que normalmente utiliza 16 bits, pueden memorizarse dos caracteres, el uno en la mitad de la izquierda (bits de 0 a 7), el otro en la de la derecha (bits de 8 a 15).
Para memorizar más de dos caracteres debe dimensionarse una matriz constituida por tantos elementos como cuantos son los pares de caracteres. Además de las enteras, a veces pueden utilizarse también otros tipos de matrices, por ejemplo de doble precisión. En estos casos, el número de caracteres que puede contenerse en un elemento es mayor.
Algunos Compiladores permiten adoptar una metodología similar a la típica del Basic utilizando un nombre de variable para indicar una cadena. En estos casos, la gestión de las cadenas es análoga a la del Basic, pero no necesita el uso del símbolo $. Por ejemplo, las siguientes asignaciones

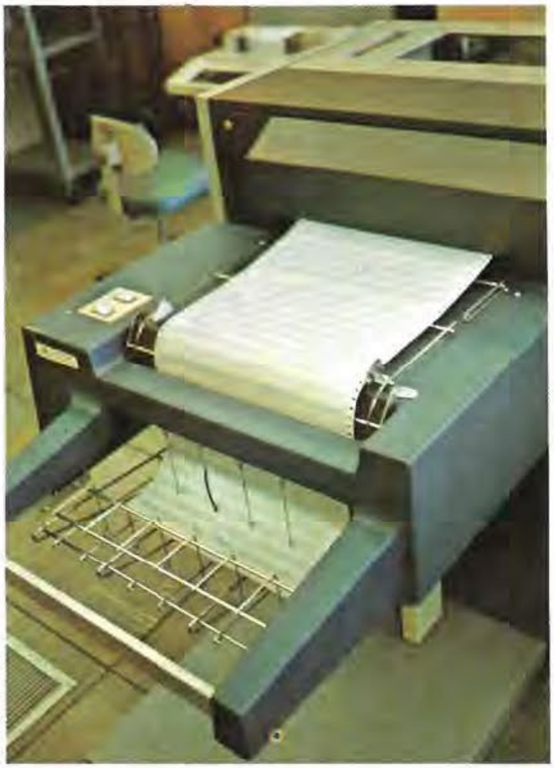
NOMBRE$ = "Prueba" (Basic)
NOMBRE = 'Prueba' (Fortran)

son idénticas.*

* En el Fortran normalmente se emplea la comilla sencilla, que sustituye a la doble.

OPERADORES LOGICOS Y RELACIONALES

Fortran	Significado	Basic	Significado	Ejemplo Basic	Ejemplo Fortran
.EQ.	Equal	=	igual	IF A = B...	IF (A.EQ.B)
.NE.	Not equal	< >	diferente (no igual)	IF A<>B...	IF (A.NE.B)
.LT.	Less than	<	menor	IF A<B...	IF (A.LT.B)
.GT.	Greater than	>	mayor	IF A>B...	IF (A.GT.B)
.LE.	Less-equal	< =	menor o igual	IF A< = B...	IF (A.LE.B)
.GE.	Greater-equal	> =	mayor o igual	IF A> = B...	IF (A.GE.B)
.NOT.		NOT	negación	NOT B	.NOT.B
.AND.		AND	AND lógico	A AND B	A.AND.B
.OR.		OR	OR lógico	A OR B	A.OR.B



Marka/Infophoto

Arrastre del módulo continuo a la salida de la impresora.

La operación de extracción de una parte de cadena requiere especificar los caracteres extremos de la sección a tomar.
Por ejemplo, volviendo a la NOMBRE = 'Prueba', la línea

NOMBRE1 = NOMBRE(2:3)

extrae y asigna a NOMBRE1, el valor «ru» y equivale a la NOMBRE1$ = MID$ (NOMBRE$ (2.2)) del Basic.
Además es posible concatenar dos cadenas con el símbolo / /. Por ejemplo, las líneas

```
N1 = 'Prueba'
N2 = 'de concatenado'
N = N1 // N2
```

asignan a la variable N el valor «prueba de concatenado». El equivalente Basic utiliza el símbolo + (N1$ + N2$) o &.
Si en Fortran se asigna a una variable el valor resultante de un cálculo cualquiera, el Compilador supone para las variables un tipo dependiente de los operandos. Por ejemplo, definiendo B y C como enteros, el cálculo A = B + C memoriza el resultado en la variable entera A. Las posibles combinaciones se indican en la página siguiente, que permitirán prevenir errores de truncado generados por declaraciones erróneas. Por ejemplo, si se multiplica un real en simple precisión por un entero, el resultado es un real en simple precisión que, si se memorizase en un entero, produciría errores de truncado. En general, para memorizar el resultado de un cálculo, debe utilizarse una variable del tipo más largo entre las de los varios operandos que intervienen en el cálculo.

Declaraciones de tipo de las variables

En Fortran, el programador puede definir el tipo de cada variable (entero, real, etc.). La sintaxis de las declaraciones de tipo es la siguiente

tipo nombre-1, nombre-2, ...

donde «tipo» es uno de los tipos de variables previstos, y «nombre-1, nombre-2, ...» son los nombres simbólicos de las variables a las que se quiere asignar el tipo especificado. Los tipos fundamentales previstos en el Fortran son:

INTEGER	Entero
REAL	Real en simple precisión
DOUBLE PRECISION	Real en doble precisión
COMPLEX	Complejo
LOGICAL	Lógico
CHARACTER	Carácter

Así, por ejemplo, las líneas

```
INTEGER A,NOMBRE,K
REAL V
```

definen como enteras las variables A, NOMBRE, K y real la variable V. Este tipo de declaración tiene la forma y las funciones análogas a las correspondientes del Basic; las

```
INTEGER A
DOUBLE PRECISION Z
```

equivalen a las

```
DEFINT A
DEFDBL Z
```

Obsérvese que en Basic no es necesaria la de-

claración de tipo real (que sin embargo existe y tiene como sintaxis DEFSNG), puesto que las variables se suponen por omisión todas reales salvo indicación contraria. En el Fortran sucede lo mismo, pero la declaración de tipo real se utiliza cuando menos para definir como reales las variables y cuyos nombres empiezan con I, J, K, L, M, N, que de otro modo el Compilador las supondría enteras.
En algunas formas de Fortran existe la posibilidad de definir cada tipo en precisión ampliada. Por ejemplo, la declaración INTEGER puede convertirse en INTEGER*2 (doble precisión) o INTEGER*4 (cuádruple precisión). INTEGER*2 declara siempre un número entero, pero con una precisión doble a la de la INTEGER (que no debe confundirse con DOUBLE PRECISION que corresponde a los reales).
La declaración CHARACTER permite definir una variable de cadena. La sintaxis es

CHARACTER*n NOMBRE

La variable indicada NOMBRE contendrá n caracteres; si n se omite, el Compilador lo supondrá igual a 1.
El resultado es similar al que se tiene en Basic escribiendo el nombre de la variable seguido del símbolo $, con la diferencia de que en este caso debe definirse la longitud de la variable como número de caracteres. En el Basic es el sistema que adecúa el espacio reservado a la cadena a medida que se varía su contenido; en el Fortran la longitud debe preverse antes.
La declaración CHARACTER puede también interesar a una matriz.
Por ejemplo, la declaración

CHARACTER*3 V(10)

define 10 cadenas de 3 caracteres cada una llamada V. En este caso ya no es necesaria la instrucción DIMENSION (dimensionado de las matrices) puesto que el espacio de memoria ya está reservado por la declaración de tipo.

TIPOS DE VARIABLES RESULTANTES DE LOS CALCULOS

PRIMER OPERANDO

SEGUNDO OPERANDO	Entero reducido	Entero	Real en simple precisión	Real en doble precisión	Complejo en simple precisión	Complejo en doble precisión
Entero reducido	ER	E	R	DP	C	CD
Entero	E	E	R	DP	C	CD
Real en simple precisión	R	R	R	DP	C	CD
Real en doble precisión	DP	DP	DP	DP	CD	CD
Complejo en simple precisión	C	C	C	CD	C	CD
Complejo en doble precisión	CD	CD	CD	CD	CD	CD

ER = Entero reducido
E = Entero
R = Real en simple precisión
DP = Real en doble precisión
C = Complejo en simple precisión
CD = Complejo en doble precisión

Las declaraciones de tipo descrita son «explícitas», puesto que cada variable debe indicarse con su propio nombre. Existe la posibilidad de definir simultáneamente el tipo de más variables agrupadas en base a la primera letra del nombre; en este caso se habla de declaraciones «implícitas». La sintaxis es la siguiente:

IMPLICIT tipo (X-Y)

donde «tipo» es uno de los anteriores (INTEGER, REAL, etc.) y X e Y indican el intervalo de letras a considerar. La declaración implícita

IMPLICIT INTEGER (A-L)

define enteras las variables cuyo nombre empieza con letras comprendidas entre la A y la L, y tiene un significado idéntico a la declaración Basic

DEFINT A-L

Algunos Compiladores Fortran prevén la opción INPLICIT NONE, que elimina todas las declaraciones implícitas; en este caso, cada variable debe declararse de modo explícito.

Dimensionado de las variables estructuradas

Las instrucciones que reservan áreas de memoria a variables dimensionadas son las mismas del Basic: DIMENSION (en Basic DIM) y COMMON.
La instrucción DIMENSION se utiliza de forma idéntica a la DIM del Basic, mientras que la COMMON, como ya se ha indicado, tiene otro significado.

Las instrucciones COMMON y EQUIVALENCE

Un programa Fortran tiene la misma estructura que un programa Basic: está constituido por un main que llama a diversas subrutinas.
La diferencia sustancial se tiene en la transferencia de las variables. En el Fortran, cada rutina tiene las suyas y no las comunica al resto del programa, salvo que dichas variables hayan sido declaradas comunes con la instrucción COMMON. En la declaración COMMON pueden dimensionarse de manera implícita variables estructuradas, como se ha visto para la declaración de tipo, con el doble efecto de hacer comunes algunas matrices y de permitir su dimensionado.

Es decir, si la variable A(20) debe ser utilizada por más puntos del programa, con la instrucción

COMMON A(20)

se asigna, al mismo tiempo, la dimensión y se impone el uso común de la matriz. En el gráfico de la página siguiente se han indicado esquemáticamente algunas situaciones típicas en el uso de las instrucciones COMMON y DIMENSION.
La instrucción COMMON puede utilizarse para transferir simultáneamente un bloque de variables identificándolo con un nombre complejo (**labelled common**). Por ejemplo:

COMMON/PRUEBA/V1,R,K,M

asigna el nombre PRUEBA al bloque de variables V1,R,K,M.
El uso de un bloque hace posible cambiar los nombres de las variables de una a otra subrutina.
Por ejemplo, si la instrucción anterior se inserta en el main y en una subrutina hay la

COMMON/PRUEBA/A1,A2,B(2)

el significado sería el siguiente

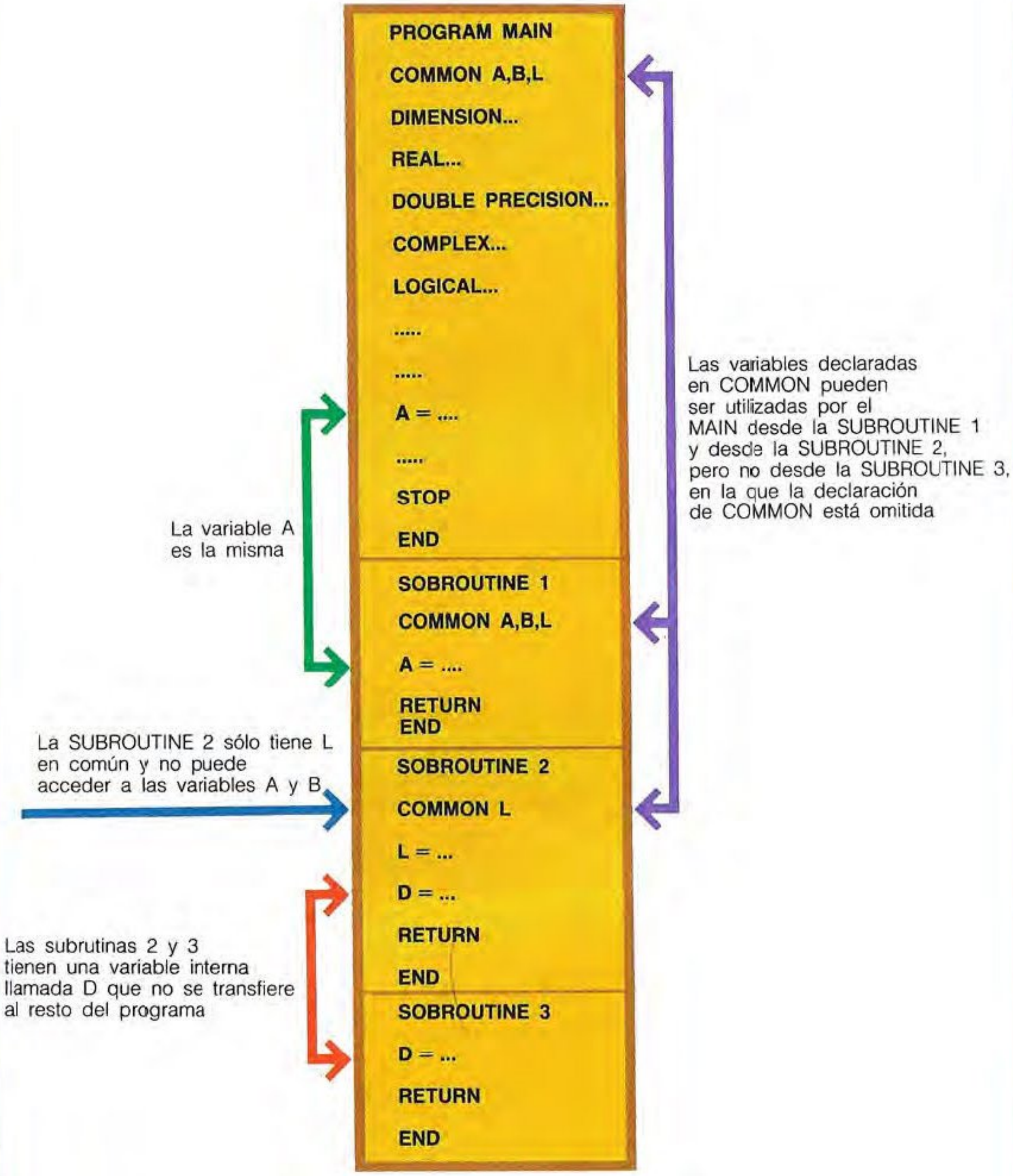
A1 = V1, A2 = R, B(1) = K, B(2) = M

Utilizar una labelled common significa identificar un área de memoria, asignarle un nombre y memorizar en ella los valores de las variables especificadas. Si en la instrucción no aparece un nombre del bloque, entonces se habla de **blank common**: la instrucción COMMON A(20) es una blank common.
En Fortran existe la posibilidad, no prevista en el Basic, de definir «equivalentes» algunos nombres de variables (incluso dimensionadas). La instrucción EQUIVALENCE (A,B) asigna simultáneamente a la misma área de memoria los dos nombres A y B. En el caso de variables dimensionadas debe verificarse que la EQUIVALENCE implique elementos contiguos. Las instrucciones

```
DIMENSION      C(3)
EQUIVALENCE    (C(1),A),(C(2),B)
```

son correctas, puesto que los elementos C(1) y

TRANSFERENCIA DE VALORES EN COMMON



C(2) son contiguos. Y viceversa, la instrucción

```
EQUIVALENCE (C(1),A),(C(3),D)
```

contiene un error, puesto que C(1) y C(3) pertenecen a la misma matriz pero no son elementos contiguos.

Instrucciones de asignación

La asignación de un valor a una variable se obtiene automáticamente utilizando el símbolo =, de manera idéntica que en el Basic:

```
V = 3.5
A = 7 + V*2
B = A*3 / (2 + 1)
```

son todas formas correctas que no presentan ninguna diferencia con respecto a las aceptadas en las versiones más recientes del Basic (recordemos que las primeras versiones del Basic preveían el código LET).
Un segundo modo de asignar valores consiste en la utilización de la instrucción DATA, que tiene un idéntico significado a la correspondiente instrucción Basic, pero se aplica con sintaxis notablemente diferente.
La instrucción Fortran

```
DATA A,R,V/3.7,4.21,5.9/
```

asigna los valores

```
A = 3.7
R = 4.21
V = 5.9
```

La lógica es la del Basic: a cada variable se asigna, progresivamente, uno de los valores numéricos elegidos, pero la forma de la instrucción es completamente diferente (el equivalente Basic sería READ A,R,V seguida de DATA 3.7,4.21,(5.9).
Más allá del aspecto formal, la principal diferencia consiste en la ausencia de la instrucción READ, que está implícita en la DATA.
En Basic, todas las variables interesadas por una asignación en DATA pueden agruparse en un punto del programa tan lejano como el que contiene los valores a asignar; la instrucción READ incrementa el puntero y extrae los valores del área DATA.
En Fortran, dada la diferente estructura de la instrucción, esto no es posible; cada variable debe aparecer en la misma instrucción en que aparece el valor a asignarle.
Por ejemplo, las instrucciones Basic

```
10 READ A,B,C.
20 READ D
30 RESTORE
40 READ E,F
50 DATA 1,2,3,4
```

se convierten en Fortran en

```
DATA A,B,C,D/1,2,3,4/
DATA E,F/1,2/
```

En la forma Basic, los valores 1 y 2 contenidos en la DATA pueden utilizarse tanto para las variables A,B como para las E,F (la línea 30 restablece el puntero). En Fortran no existe el equivalente del código RESTORE y los valores numéricos deben repetirse necesariamente.
Para asignar los valores a las variables dimensionadas (matrices) puede usarse una forma de bucle implícito. Por ejemplo, para asignar los valores a los elementos de la matriz V(3) (dimensionadas a 3) pueden usarse las dos formas:

```
DATA V(1),V(2),V(3)/11.6,7.9,3.1/
DATA (V(I),I = 1,3)/11.6,7.9,3.1/
```

Obsérvese que en Fortran no existe ninguna posibilidad de dimensionado implícito en la fase de asignación. Mientras que en Basic el ejemplo indicado podría utilizarse directamente sin dar error (la variable V se dimensionaría automáticamente), en Fortran es necesaria una instrucción previa: DIMENSION V(3).
Por tanto, la forma completa es

```
DIMENSION V(3)
DATA (V(I),I = 1,3)/11.6,7.9,3.1/
STOP
END
```

En la instrucción DATA puede insertarse cualquier tipo de constante, siempre que sea homogénea con el tipo de la variable asociada. Por ejemplo, las instrucciones

```
CHARACTER CH
DATA CH/'A'/
```

definen CH como variable que contendrá carac-

teres y le asignan el carácter A (en la DATA los caracteres deben ir encerrados entre comillas). Una opción que se encuentra frecuentemente en la instrucción DATA es el símbolo * con el significado de repetición. Por ejemplo, la línea

```
DATA (V(I),I = 1,3)/3*5/
```

asigna el valor 5 a los tres elementos de la matriz V(3): la simbología 3 * significa «tres veces el valor que sigue».
Para algunos Compiladores Fortran, el bucle implícito puede omitirse; en este caso, la instrucción anterior también puede escribirse así:

```
DATA V/3*5/
```

siempre después de haber definido la variable V como matriz de 3 elementos [DIMENSION V(3)].
Resumiendo, las cuatro formas principales de DATA se ilustran en la tabla de abajo.
El último modo para asignar valores numéricos, característicos del Fortran, consiste en emplear la instrucción PARAMETER.
Esta instrucción sirve para definir valores constantes con un nombre simbólico. La constante puede ser de cualquier tipo, aunque homogénea con el tipo asociado al nombre (por ejemplo, constantes y enteras con nombres que empiezan con I,J,K,L,M,N). La sintaxis es:

```
PARAMETER (nom.-1 = const.-1, nom.-2 =...)
```

La instrucción

```
PARAMETER (PI = 3.14)
```

asigna al nombre simbólico PI el valor 3.14. En este caso no es necesaria ninguna declaración de tipo, puesto que PI es un nombre al cual puede asociarse un valor real (3.14).
En el caso de que no exista compatibilidad, antes de la PARAMETER debe definirse el tipo de variable. Por ejemplo:

```
CHARACTER * 6 D
PARAMETER (D = 'PRUEBA')
```

define D como una cadena de 6 caracteres y le asigna el valor PRUEBA.

Las instrucciones del Fortran

Ahora examinaremos las principales instrucciones del Fortran, evidenciando las que difieren del conjunto de las instrucciones previstas en el Basic y ya descritas en su momento.
La notable similitud que existe entre los dos lenguajes permite simplificar la exposición, limitándola a la enumeración de las únicas diferencias sustanciales.

La instrucción GOTO

En Fortran existen tres tipos de GOTO*:

- GOTO incondicionado
- GOTO calculado
- GOTO asignado

La forma incondicionada es idéntica a la utiliza-

* La presencia de un espacio entre las palabras GO y TO no es significativa, como sucede en Basic.

FORMAS DE LA INSTRUCCION DATA

CHARACTER CH DATA CH / 'A' /	Declaración de tipo DATA variable, variable, ... / valor, valor, ... /
DIMENSION V(5) DATA (V(I),I = 1,3) / 4.2,7.1,8.3 /	Dimensionado DATA bucle / valores /
DIMENSION V(5) DATA (V(I),I = 1,3) / 3*5.1 /	Dimensionado DATA bucle / factor de repeticición*valor /
DIMENSION V(5) DATA V / 5*3.7 /	Dimensionado DATA variable dimensionada / factor de rep.*valor /

da en Basic. También en Fortran debe indicarse el salto (GOTO) a una línea identificada por un número; por esto, la sintaxis es

GOTO número de línea

El GOTO calculado, a pesar de tener la equivalencia en Basic, tiene una forma diferente. En Basic, la instrucción es ON K GOTO...; en Fortran, la sintaxis es:

GOTO (núm.-de-línea-1, núm.-de-línea-2, ...), K

El significado es el siguiente: para K = 1 salta al primer número de línea especificado (núm.-de-línea-1), para K = 2 al segundo, etc.
La forma exacta prevé la coma antes del parámetro K, pero en general puede omitirse. El índice (parámetro K) también puede ser el resultado de un cálculo, como por ejemplo la

GOTO (100,10,1240),R + 4

activa un salto a una de las instrucciones 100,10,1240 en base al resultado del cálculo R + 4 (o mejor dicho, en base al valor de la parte entera de este resultado).
La tercera forma, GOTO asignado, no encuentra equivalencia en el Basic; permite utilizar un nombre simbólico en lugar de un número de línea y debe ir precedido de una instrucción que asigne a la línea el nombre simbólico.
La instrucción GOTO 150 puede convertirse así en GOTO LLEGADA, habiendo asignado a la línea 150 el nombre LLEGADA. La instrucción que permite efectuar esta última asignación es

ASSIGN número-línea TO nombre

En el caso del ejemplo anterior se tiene

```
ASSIGN 150 TO LLEGADA
....
GOTO LLEGADA
```

Al ejecutar la instrucción GOTO LLEGADA, el programa transfiere el control a la línea numerada con 150.

Los bucles

La ejecución de un bucle sigue las reglas vistas en la exposición del Basic, pero en Fortran la sintaxis es diferente:

DO n índice = valor inicial, valor final, paso

DO es el código que activa el bucle (equivalente al FOR), «n» es el número de línea en que termina el bloque de instrucciones a realizar repetidamente en el bucle, «índice» es el índice del bucle que varía del «valor inicial» al «valor final» con incrementos iguales a «paso». Si el valor del paso se omite, el Compilador supone por omisión que es 1.
La instrucción:

DO 100 I = 1,22,3

activa la ejecución repetitiva de la parte del programa comprendida entre esta línea y la numerada con 100, haciendo variar el índice I entre los valores 1 y 22 con paso 3.
La línea que termina el bucle no debe contener ninguna referencia al índice (en Basic hace falta la instrucción NEXT I), y podría ser una instrucción Fortran cualquiera.
En realidad, se prefiere terminar los bucles con una instrucción CONTINUE. Esta instrucción no tiene ningún efecto en el programa (significa literalmente «continúa») y tiene la única finalidad de servir de apoyo al número de línea, evidenciando el final del bucle. El bucle anterior, completo, se convierte en

```
    DO 100 I = 1,22,3
    ...... ┐
    ......  │ Instrucciones ejecutadas
    ......  } repetitivamente
    ...... ┘
100 CONTINUE
```

En algunas formas de Fortran, en lugar del código CONTINUE (que ya está previsto), se adopta el código (más descriptivo) END DO.
En este caso, el programa se convierte en

```
    DO 100 I = 1,22,3
    (parte repetitiva)
100 END DO
```

Bucles implícitos. En Fortran es posible activar bucles de modo implícito omitiendo la instrucción DO. Se ha presentado un ejemplo que ilustra la asignación con el DATA aplicada a una matriz. En general, un bucle implícito (normalmente llamado DO implícito) tiene la forma:



IBM

Perforador de fichas IBM 3525. Es uno de los periféricos estándar de salida de ordenador.

XX (matriz, índice = val-inicial, val-final, paso)...

La parte central de la instrucción encerrada entre paréntesis es el bucle propiamente dicho, la sigla XX representa una de las instrucciones que admite este tipo de bucle, y el punteado final indica una eventual continuación. En el caso de la asignación con la DATA, la parte indicada con XX está sustituida por DATA, y la continuación de la instrucción está constituida por valores numéricos a asignar (/11.6,7.9,.../).
Las instrucciones que soportan el bucle implícito sólo son de dos tipos:

— DATA
— Instrucciones I/O

Para imprimir el contenido de una matriz con 10 elementos puede adoptarse la instrucción

```
WRITE (p,n) (V(I),I = 1,10)
```

WRITE es el código de la instrucción (equivalente a PRINT del Basic), «p» y «n» son parámetros que especifican dónde (en qué periférico) y cómo (con qué formato) escribir, la expresión entre paréntesis indica las variables a escribir.
El DO implícito es formalmente similar a un FOR... NEXT... desarrollado sobre una sola línea. Por ejemplo, para escribir los elementos de la matriz anteriormente considerada, la forma Basic podría ser:

```
FOR I = 1 TO 10:PRINT V(I):NEXT I
```

que es el equivalente de la instrucción Fortran

```
WRITE (p,n) (V(I),I = 1,10)
```

Los DO implícitos pueden ser anidados, es decir contenidos uno dentro del otro. La impresión de los elementos de una matriz de dos dimensiones puede obtenerse, por ejemplo, con la instrucción

```
DIMENSION A(5,3)
........
WRITE (p,n) ((A(I,J),J = 1,3),I = 1,5)
```

que en Basic tiene el equivalente

```
10 FOR I = 1 TO 5
20 FOR J = 1 TO 3
30 PRINT A(I,J)
40 NEXT J
50 NEXT I
```

La forma DO... WHILE... Este tipo de bucle no está previsto en el Fortran ANSI 77, y sólo está presente en algunas máquinas. Sin embargo, lo mencionaremos porque es muy similar a la WHILE... WEND del Basic.
La sintaxis de la instrucción es

```
DO n WHILE (condición)
```

donde «n» es el número de línea que termina el DO y «condición» es una condición que, si se cumple, produce la repetición del ciclo.
Si se introduce la instrucción

```
DO 100 WHILE (A.GT.B)
```

el bucle se repite hasta que (WHILE) el valor de la variable A es mayor (.GT.) que el valor de B. Normalmente, el número de línea puede omitirse, y el final del DO se indica con END DO. En este caso, el ejemplo anterior se convierte:

```
DO WHILE (A.GT.B)
......
......
END DO
```

sin ningún número de línea.

Instrucciones condicionadas

El condicionado de una instrucción se obtiene, como en Basic, anteponiendo el código IF. En Fortran existen tres tipos principales de IF:

- IF aritmético
- IF lógico
- IF de bloques

Los primeros dos tipos son comunes a todas las versiones del lenguaje, y el último sólo es característico de los más modernos.
La sintaxis del IF aritmético es la siguiente:

```
F (expresión) línea-1, línea-2, línea-3
```

La expresión se evalúa y el control se transfiere a la correspondiente línea según el signo del resultado. Por ejemplo, ejecutando la instrucción

```
IF (A + B) 10,100,1000
```

el programa salta a la línea 10, a la 100 o a la 1000 en base al resultado (negativo, nulo o positivo) de la suma A + B.
La instrucción debe contener sólo 3 direcciones, puesto que la selección se obtiene en base al signo: si el resultado es negativo, el control pasa a la primera línea citada, si es 0 a la segunda y si es positivo a la tercera. La lógica de ejecución se ilustra en el gráfico de la página siguiente.
El IF aritmético del Fortran equivale a tres IF lógicos en serie. Por ejemplo, la instrucción

```
IF K 10,20,30
```

significa

```
IF (K.LT.0) GOTO 10
IF (K.EQ.0) GOTO 20
IF (K.GT.0) GOTO 30
```

Obsérvese la exacta correspondencia de estos IF (aparte de la simbología .LT., .EQ.) con los correspondientes Basic (IF K = 0 GOTO 10).
En el IF aritmético, los números de línea pueden repetirse:

IF K 10, 10, 10 transfiere en cualquier caso el control a la línea 10
IF K 10, 20, 10 transfiere el control a la línea 20 sólo si K = 0
IF K 10, 10, 20 transfiere el control a la línea 20 sólo si K > 0

A diferencia del IF aritmético, el IF lógico tiene las mismas funciones que el correspondiente Basic. La sintaxis es

```
IF (condición) instrucción
```

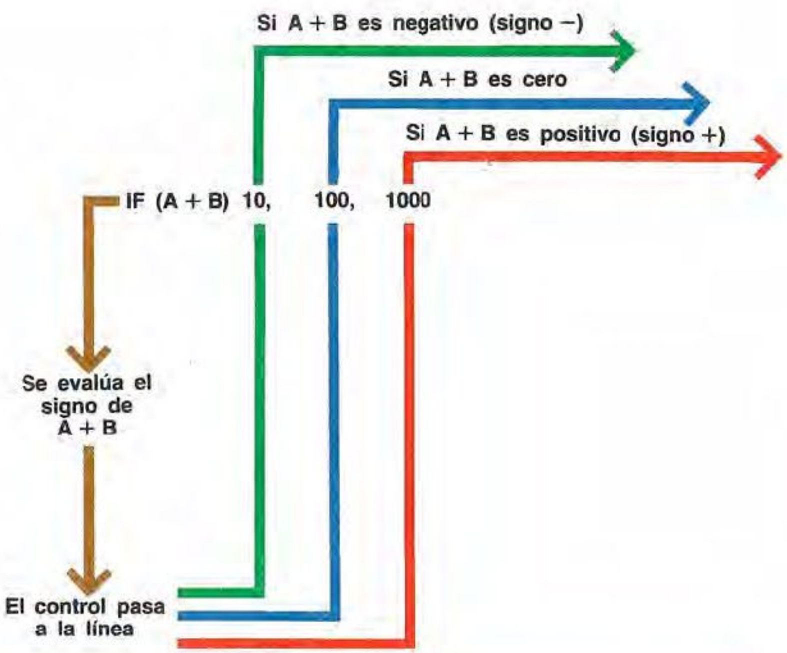
La «instrucción» sólo se ejecuta si la «condición» es verdadera. La única diferencia con el Basic es la ausencia del código THEN: la instrucción Basic

```
IF A > B THEN K = 1
```

se convierte en

```
IF (A.GT.B) K = 1
```

LOGICA DE EJECUCION DE UN IF ARITMETICO



La «instrucción» puede ser de cualquier tipo, excepto DO, END u otro tipo de IF lógico. El IF tiene bloques y una extensión del IF lógico que permite activar bloques de instrucción en lugar de la única instrucción prevista en dicho IF lógico. La sintaxis es

```
IF (expresión) THEN
..............
bloque de instrucciones a realizar si
la expresión es verdadera
..............
END IF
```

Obsérvese que normalmente END IF está escrito sin espacios (ENDIF), pero no se trata de un vínculo; algunas versiones aceptan las dos palabras separadas.

En el IF de bloques aparece la palabra THEN como en el Basic, y es posible utilizar la especificación ELSE (de otro modo). Por ejemplo, la secuencia de instrucciones

```
IF (A.EQ.B) THEN
C = 3 * K
D = C + 2
ELSE IF (A.GT.B) THEN
C = 0
D = 1
ENDIF
```

asigna los valores C = 3 * K y D = C + 2 bajo la condición A = B, mientras que para A > B asigna C = 0 y D = 1. Las especificaciones ELSE pueden estar contenidas en un número cualquiera dentro de un mismo IF; por tanto, se dispone de un medio muy potente de selección, aunque no es demasiado sencillo de aplicar.

El lenguaje Fortran, en particular la versión ANSI 77, es mucho más estructurado que el Basic, y necesita un período de adaptación por parte del programador. En cambio, el programa es más documentado, y en ciertas condiciones puede modificarse con mayor facilidad.

El tema de la programación estructurada volverá a abordarse más adelante, cuando se ilustre el lenguaje Pascal, que nació precisamente para permitir una fácil programación estructurada.

El uso de la cláusula ELSE en el interior de un IF se ha previsto también en algunas formas de Basic, pero, salvo casos particulares, no es tan elástico como en el Fortran.

Uso de las subrutinas

En la forma más sencilla, las subrutinas Fortran son gestionadas de manera similar a la vista en Basic, aunque se uniformizan con una filosofía completamente diferente, al menos con respecto al Basic estándar. En Fortran, cada subrutina es una entidad en sí misma que no comunica implícitamente con las otras partes del programa. En cambio, en Basic, la subrutina constituye una parte del programa que puede utilizarse repetidamente, y como tal comparte con el programa principal todas las variables.
La particular estructura de las subrutinas Fortran puede constituir, en ciertas condiciones, una ventaja. En un programa particularmente largo, y por tanto muy fraccionado, puede ser útil la segregación de las subrutinas como entidades en sí mismas. Los nombres de las variables pueden reutilizarse con significados diferentes, lo que conduce al programador a la necesidad de recordar los nombres y las condiciones de cada variable.
La sintaxis completa de la instrucción de llamada de una subrutina es la siguiente:

```
CALL nombre (var-1, var-2, ..., * n1, * n2)
```

donde «nombre» es el nombre simbólico dado a la subrutina (punto de entrada, equivalente al número de línea del Basic); «var-1, var-2, ...» son eventuales variables que deben transferirse a la subrutina y de ésta al programa llamador; «n1, n2...» son los puntos de retorno alternativos previstos.
Efectivamente, las versiones Fortran más completas prevén la posibilidad de retorno de una subrutina a una línea diferente de la que ha sido llamada.
Por ejemplo, la instrucción

```
CALL PRUEBA (* 100, * 200, * 300)
```

activa el proceso de retorno parametrizado. Según la particular salida seleccionada en la subrutina PRUEBA, el control se transfiere a una de las tres líneas especificadas (100, 200, 300) del main. Naturalmente, las tres instrucciones RETURN insertadas en la subrutina deben citar el parámetro.
En particular, en la rutina PRUEBA deberían estar previstos por lo menos tres retornos: RETURN 1 para la línea 100, RETURN 2 para la 200 y RETURN 3 para la 300.
Este tipo de RETURN puede parametrizarse. Por ejemplo, escribiendo RETURN K se tiene el retorno a 100, 200, 300 en función del valor de K, que puede calcularse en la misma subrutina. Si este valor no está comprendido dentro de los límites previstos, el control vuelve a la línea que sigue a la llamada (sin embargo, esta posibilidad no es común a todas las versiones existentes del Fortran).
La primera línea de la subrutina (SUBROUTINE...) debe contener las referencias a los retornos alternativos en número igual a cuantos sean los retornos previstos. Por ejemplo, la llamada anterior necesita una primera línea del tipo

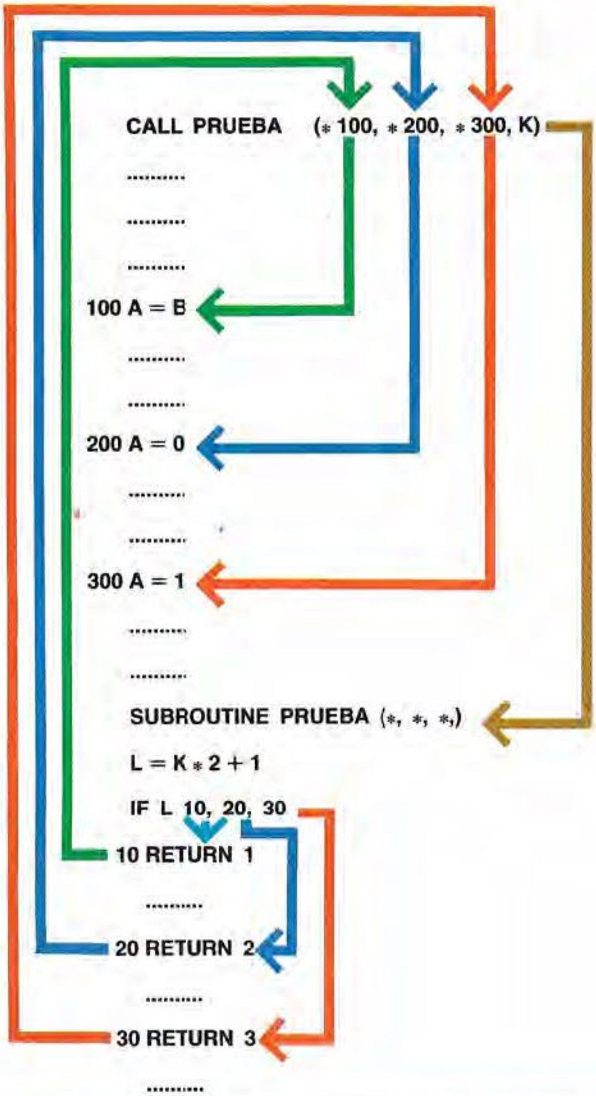
```
SUBROUTINE PRUEBA (*, *, *)
```

puesto que debe haber previstos tres retornos alternativos.
En el gráfico de la página siguiente se ha indicado un ejemplo de la lógica expuesta. El punto de reentrada está determinado por el valor que asume L (resultado de un cálculo).
En las subrutinas Fortran, los nombres de las variables transferidas como parámetros no deben coincidir necesariamente. Si una subrutina realiza la suma de dos números (A + B) y coloca el resultado en una tercera variable (C), deberá llamarse relacionando tres parámetros: los dos sumandos y el resultado. Su primera línea será del tipo SUBROUTINE SUMA (X, Y, R), indicando con X e Y los sumandos y con R el resultado.
En la llamada pueden utilizarse otros nombres, siempre que se respete el orden; en cualquier caso, los dos primeros indicarán los sumandos y el tercero el resultado.
En el programa

```
      PROGRAM PRUEBA
      A = 1
      B = 2
   10 CALL SUMA (A,B,S)
      .......
      .......
      X = 3
      Y = 5
   20 CALL SUMA (X,Y,T)
      .......
      SUBROUTINE SUMA (X,Y,R)
      R = X + Y
      RETURN
```

la línea 10 (las numeraciones 10 y 20 no son necesarias a los fines del programa, y se han utilizado sólo para facilitar su lectura) llama la

USO DE LAS REENTRADAS ALTERNATIVAS



subrutina SUMA con los parámetros A,B,S; al retorno, la suma está en S. La línea 20 llama la misma rutina con los parámetros X,Y,T, y al retorno la suma está en T.

Es decir, el significado de cada parámetro sólo depende de su posición. En la subrutina se ha realizado el cálculo R = X + Y, es decir, se ha calculado el tercer parámetro como suma de los otros dos. Cualquiera que sea el nombre simbólico de los parámetros, al retorno de la subrutina, la tercera variable contiene siempre la suma de las dos primeras.

Instrucciones de terminación y de paro de la ejecución

En este párrafo examinaremos tres instrucciones, una de las cuales no tiene correspondencia en Basic, mientras que las otras dos tienen en Fortran un significado diferente del que tienen en Basic. Las instrucciones son

```
PAUSE
STOP
END
```

La instrucción PAUSE sirve para suspender momentáneamente la ejecución del programa, por ejemplo para dar tiempo al operador de realizar eventuales operaciones fuera de línea, como colocar el papel en la impresora. La sintaxis de la instrucción es

PAUSE n

o bien

PAUSE 'mensaje'

Entre los dos casos, el sistema, antes de suspender la ejecución del programa, avisa que ha encontrado la pausa «n» o «mensaje». La ejecución de la instrucción: PAUSE 'Alinear el papel' genera la presentación del mensaje «PAUSE Alinear el papel».

El programa detenido puede reactivarse con una oportuna instrucción cuyo código depende de la máquina; el más usado es GO. Introduciendo este comando (por teclado), el sistema prosigue la ejecución a partir de la línea siguiente a la PAUSE.

La instrucción END sirve para señalar al Compilador el fin de cada «unidad de programa». El Fortran es más estructurado que el Basic estándar, y cada subrutina es una unidad compilada separada de las otras. Por tanto, pueden encontrarse diferentes líneas END en el mismo programa; sólo una corresponde al main e indica el fin lógico del proceso, mientras que las otras indican el fin de subrutinas.

Normalmente, el código END indica en una subrutina las mismas funciones que el RETURN, pero no es verdad lo contrario: omitiendo la palabra END al final de una subrutina se tiene error durante la compilación.

El STOP tiene el mismo efecto que en el Basic: detiene la ejecución del programa.

En Fortran puede asociarse a un mensaje o a un valor numérico; uno aparece en el vídeo al producirse el STOP.

Obsérvese que la PAUSE y el STOP pueden contener un mensaje, pero deben utilizar por lo menos un valor numérico; generalmente no se admite el uso de esta instrucción sola.

Las funciones del Fortran

En el Fortran, como en el Basic, existen las funciones intrínsecas (de librería) y las definidas por el usuario.

La librería del Fortran es más rica en funciones matemáticas que la del Basic, mientras que dispone de poquísimas funciones para el proceso de las cadenas. Esta particular estructura procede del destino original preferentemente científico del Fortran. Sin embargo, la evolución paralela de los dos lenguajes los ha aproximado notablemente, hasta el punto que existen formas de Fortran que prevén las mismas funciones de cadena que el Basic, así como el propio Basic posee implantaciones que disponen de todas las funciones matemáticas del Fortran.

A continuación se relacionarán algunas de las funciones principales que pueden utilizarse en Fortran. Su variedad depende de la versión particular del Compilador; por tanto se presentarán sólo las principales funciones previstas en la norma ANSI 77.

Funciones de librería

El Fortran es muy rico en funciones de librería orientadas al desarrollo de cálculos. Las versiones más recientes incluyen también una discreta variedad de funciones de cadena, que permiten utilizar asimismo este lenguaje en todas las aplicaciones. A continuación, las funciones previstas en la versión ANSI 77 se dividirán por grupos homogéneos. Esta subdivisión no tiene el carácter de una clasificación propiamente dicha; sólo es un medio para facilitar la lectura y los eventuales paralelos con el Basic.

Dado que la comprensión y la utilización de algunas funciones matemáticas requieren una preparación específica, estas funciones particulares sólo se indican para dar a conocer su existencia. Su aprendizaje no es una materia de la técnica de programación y puede dejarse sin alterar el conocimiento del lenguaje.

Conversión de tipo. A este grupo pertenecen las funciones que permiten modificar el tipo de una variable. A continuación se indican con una ulterior subdivisión en función del tipo de variable generada.

Las funciones de **conversión en entero** permiten transformar en entero el valor de variables reales o en doble precisión. La sintaxis para la transformación del valor de una variable de real a entero es la siguiente:

E = INT(R)
E = IFIX(R)



Sistema de proceso de Intel Corporation que puede ejecutar comandos vocales.

(R = real, E = entero)

mientras que para transformar en entero un valor de doble precisión debe escribirse

E = IDINT(R)

Las funciones Basic equivalentes son INT (o CINT) y FIX.

Las funciones de **conversión en real** (simple precisión) tienen la sintaxis

de entero a real R = FLOAT(E)
de doble precisión a real R = SNGL(D)
(R = real, E = entero, D = doble precisión)

El equivalente Basic es CSNG.

La **conversión en doble precisión** tiene una sintaxis única cualquiera que sea el tipo del argumento:

D = DBLE(X) (X = entero, real, complejo)

El equivalente Basic, excepto en el caso de X complejo, es CDBL(X).

La **conversión carácter/número** en cambio se obtiene con la siguiente sintaxis

de carácter a entero E = ICHAR(C)
de entero a carácter C = CHAR(E)
(E = entero, C = carácter)

El valor de I es equivalente al carácter según el código ASCII.
Las funciones equivalentes en Basic son: ASC y CHR$.

Para la **conversión de un real en complejo**, la función a utilizar es CMPLX(R) y genera un número complejo con parte real igual a R y parte imaginaria igual a 0.
No tiene correspondiente en Basic.

Funciones de cálculo. Están orientadas a la realización de determinados cálculos aritméticos de forma inmediata.

Para el **truncado de la parte entera** del valor de una variable se utilizan las funciones

A = AINT(B) A y B reales en simple precisión
C = DINT(D) C y D en doble precisión

En cambio, para calcular el **entero más grande** contenido en el valor de una variable se utilizan las siguientes:

A=ANINT(B) A y B reales en simple precisión
C=DNINT(D) C y D en doble precisión
E = NINT(B) E entero, B real en simple precisión
E = IDINT(D) E entero, D en doble precisión

El **valor absoluto** de una variable, o sea el valor sin signo (de la parte real si el argumento es complejo), se obtiene con las funciones

E = IABS(K) E y K enteros
A = ABS(B) A y B reales en simple precisión
C = DABS(D) C y D en doble precisión
C = CABS(Z) C real, Z complejo

La función de **transferencia del signo** necesita dos argumentos, A y B, y restituye – A si B es menor de cero y A si B es mayor o igual a cero. Es preciso que los argumentos sean del mismo tipo, y la sintaxis es diferente según el tipo de los argumentos:

ISIGN(A,B) argumentos enteros
SIGN(A,B) argumentos reales en simple precisión
DSIGN(A,B) argumentos en doble precisión

También es necesario que la función **diferencia positiva** tenga dos argumentos (A,B) y restituye A – B si A es mayor que B y 0 si A es menor o igual a B:

IDIM(A,B) argumentos enteros
DIM(A,B) argumentos reales en simple precisión
DDIM(A,B) argumentos en doble precisión

La función **producto en doble precisión** realiza el producto entre dos números reales transformando el resultado en doble precisión:

D = DPROD(A,B) A y B reales en simple precisión
D en doble precisión

La función **resto** realiza el cálculo

A – INT (A / B) * B

proporcionando así el resto de la división entre los dos argumentos A y B.
Es muy similar al equivalente Basic, con alguna diferencia en la nomenclatura:

MOD para argumentos enteros
AMOD para argumentos reales
DMOD para argumentos en doble precisión

Las funciones **máximo** y **mínimo** permiten extraer, respectivamente, el valor máximo y el valor mínimo de una lista de argumentos:

R = MAX 0 (A,B,C...) argumentos enteros
R = AMAX (A,B,C...) argumentos reales en simple precisión
R = DMAX (A,B,C...) argumentos en doble precisión
R = MIN 0 (A,B,C...) argumentos enteros
R = AMIN (A,B,C...) argumentos reales en simple precisión
R = DMIN (A,B,C...) argumentos en doble precisión

También existen formas que permiten el uso de argumentos de tipo mixto.

Funciones de cadena. En el Fortran ANSI 77 existen sólo dos funciones para el tratamiento de las cadenas:

LEN proporciona la longitud (en caracteres) de una cadena
INDEX restituye la posición inicial de una subcadena en el interior de una determinada cadena

Las **funciones lexicales** permiten confrontar dos cadenas (siguiendo el ordenamiento alfabético) y restituyen el valor «verdadero» o «falso» según que la condición de comparación se verifique o no. Las funciones previstas son:

C = LGE(A,B) verifica si A > = B
C = LGT(A,B) verifica si A > B
C = LLE(A,B) verifica si A < = B
C = LLT(A,B) verifica si A < B

En el retorno se tiene en C el valor lógico TRUE o FALSE.

Funciones matemáticas. Por razones de brevedad presentaremos en este párrafo sólo las formas principales, referidas a argumentos reales; para los argumentos en doble precisión, la mayoría de veces es suficiente sustituir la primera letra de la función por la letra D. Sin embargo, no se trata de una regla general, y para conocer la sintaxis exacta es necesario consultar el manual de la versión particular del Fortran utilizada.

Descripción	**Función**	**Ejemplo**
Raíz cuadrada	SQRT	B=SQRT(R)
Exponencial	EXP	B=EXP(R)
Logaritmo natural	LOG	B=LOG(R)
Logaritmo decimal	LOG10	B=LOG10(R)
Seno	SIN	B=SIN(A)
Coseno	COS	B=COS(A)
Tangente	TAN	B=TAN(A)
Arcoseno	ASIN	B=ASIN(R)
Arcocoseno	ACOS	B=ACOS(R)
Arcotangente	ATAN	B=ATAN(R)

Las funciones trigonométricas utilizan los ángulos (en los ejemplos indicados con la variable A) expresados en radianes. En el Fortran se han previsto además las funciones hiperbólicas, cuyos nombres se obtienen añadiendo la letra H al correspondiente nombre trigonométrico (por ejemplo, SINH indica el seno hiperbólico).

Funciones definidas por el usuario

Las funciones definidas por el usuario siguen bajo ciertos aspectos reglas análogas a las vistas en el Basic. Sin embargo, permanece una diferencia de fondo debido al hecho de que en Fortran cada módulo (subrutina o función) está separado lógicamente del resto del programa.
La sintaxis que permite definir una función de usuario es la siguiente:

tipo FUNCTION nombre (lista de argumentos)

tipo	permite definir el tipo de la función (real, entera, etc.) y puede asumir uno cualquiera de los valores previstos en las asignaciones (REAL, INTEGER, etc.)
FUNCTION	es la palabra reservada que indica la función
nombre	es el nombre asignado a la función; a diferencia del Basic, en que el nombre de una función está constituido por un solo carácter (DEF FN A), en Fortran puede usarse un nombre cualquiera, al igual que los de las variables
lista de argumentos	está formada, como en el Basic, por los argumentos que la función debe utilizar.

Por ejemplo, la línea

```
INTEGER FUNCTION PRUEBA ( )
```

define la función entera PRUEBA, sin argumentos; la línea

```
FUNCTION X(A,B,C)
```

define la función X, que utiliza los argumentos A,B,C; finalmente, la

```
CHARACTER * 4 A(L)
```

define la función de cadena A, de 4 caracteres de longitud y con el argumento L.
En Fortran, las funciones definidas por el usuario tienen una estructura similar a la de una subrutina; pueden desarrollarse sobre más líneas, incluyendo diversos cálculos, al contrario de lo que sucede en el Basic, donde se utiliza una sola línea con las consiguientes limitaciones.
Una función Fortran empieza con la declaración FUNCTION y prosigue sobre un número cualquiera de líneas, para terminar con la palabra RETURN.
Por ejemplo, la función

```
FUNCTION HIP(A,B)
HIP = SQRT((A * * 2) + (B * * 2))
RETURN
END
```

calcula la hipotenusa de un triángulo rectángulo de catetos A y B. Para utilizarla basta con reclamarla con el nombre simbólico (HIP).
Las líneas

```
......
A = 3
B = 4
```

HIPOTENUSA = HIP(A,B)

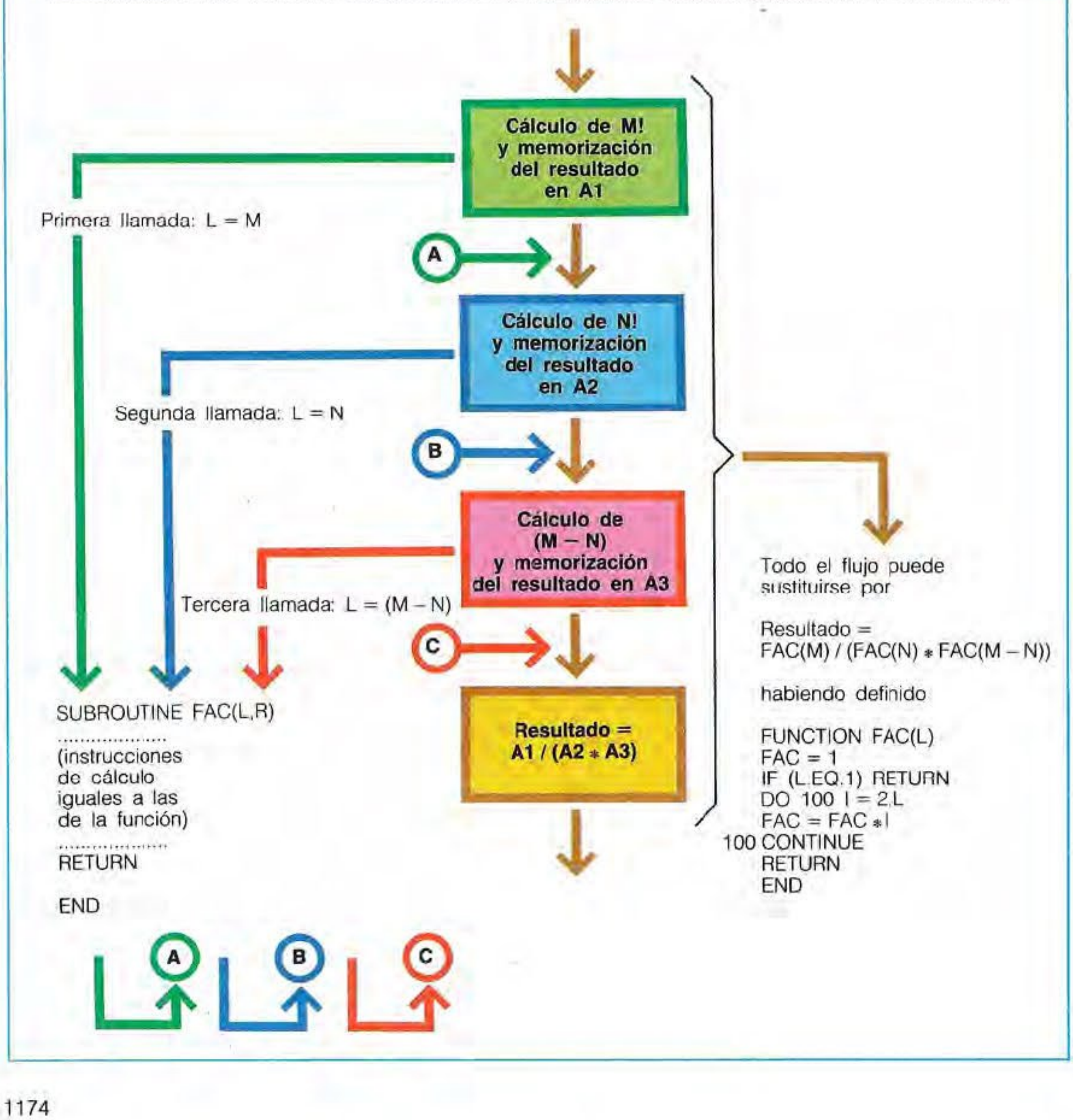
......

llaman a la función anteriormente definida y transfieren el resultado (hipotenusa) a la variable HIPOTENUSA.
Se habría obtenido el mismo resultado definido HIP como subrutina (en este caso deben citarse tres parámetros: los catetos A, B como parámetros de entrada e HIPOTENUSA como parámetro de salida), pero en general, el uso de las funciones hace el programa más legible y mejor estructurado. En el gráfico de abajo se ha mostrado el uso de una función comparado con el caso en que se utiliza una subrutina. El programa, del cual se han omitido las partes iniciales y finales, debe calcular las combinaciones de M elementos en clase N (o sea tomados en grupos de N). La fórmula a desarrollar es

$$\text{Combinaciones} = \frac{M!}{N! \times (M-N)!}$$

En general, el símbolo K! (K factorial) indica el producto de todos los números enteros a partir de 1 hasta K (por ejemplo para K = 5 se tiene K! = 1 × 2 × 3 × 4 × 5).
Si para calcular los tres factoriales se utiliza una subrutina, esta última debe llamarse tres veces,

DESARROLLO DE UN CALCULO UTILIZANDO UNA FUNCION DE SALIDA



memorizando cada vez la respuesta, como se muestra en el diagrama de flujo. La subrutina deberá tener dos parámetros, L y R. El primero es el número del que se quiere calcular el factorial, y el segundo es el resultado del cálculo. En la primera llamada se realiza el cálculo de M! (que después se memoriza, por ejemplo, en A1), en la segunda el cálculo de N! (memorizado en A2) y en la tercera llamada el cálculo de (M – N)! (memorizado en A3). El último paso es el cálculo de las combinaciones [A1/(A2 * A3)].
Utilizando una función en lugar de una subrutina, el conjunto se reduce a una sola expresión. El desarrollo de la función es idéntico a la subrutina, pero se eliminan las llamadas y las consiguientes memorizaciones intermedias.
Obsérvese el método de llamada.
En la primera línea de la función [FUNCTION FAC(L)] aparece el argumento L, que, como para las subrutinas, sólo tiene el objeto de indicar la existencia de un parámetro. Sustituyendo en la llamada el símbolo L con el nombre de una variable cualquiera o con un valor numérico, se activa el cálculo correspondiente.
Así, para calcular M! basta con llamar la función transfiriendo M [FAC(M)]; para N! se transfiere N [FAC(N)], etc.
Además de la forma ilustrada, en Fortran se han previsto también funciones que se realizan sobre una sola línea. Son más similares a las del Basic pero tienen la limitación de ser utilizables sólo en el contexto en que están definidas (las otras pueden utilizarse, al igual que las subrutinas, en cualquier punto del programa). La sintaxis de este tipo de función es la siguiente:

nombre (argumentos) = expresión

donde

nombre	es el nombre simbólico de la función
argumentos	son los parámetros que utiliza la definición
expresión	describe el cálculo a realizar

Por ejemplo, el cálculo de la hipotenusa de un triángulo rectángulo puede realizarse con una función del tipo

HIP(A,B) = SQRT((A * * 2) + (B * * 2))

Para utilizarla basta con llamarla del mismo modo visto para las demás funciones:

HIPOTENUSA = HIP(A,B)

Las instrucciones y los formatos de I/O

En Fortran, la gestión de las funciones de I/O es notablemente diferente del Basic. En el Basic estándar existe un solo periférico de entrada, el teclado (en este punto no se han considerado las funciones I/O en disco), y dos unidades de salida, o sea el vídeo y la impresora. Para cada unidad existe una instrucción dedicada, y por tanto no es necesario especificar la unidad de I/O interesada: si la instrucción es de entrada, se direcciona automáticamente al teclado; si es de salida, el código PRINT la direcciona al vídeo, el código LPRINT a la impresora. En cambio, en Fortran debe especificarse para cada operación de I/O cuál es la unidad interesada.
Esta diferencia se debe a los particulares orígenes del lenguaje. El Fortran nació en grandes procesadores, que controlan en general varias impresoras y varios dispositivos de entrada y, en su evolución, el lenguaje ha conservado sus características originales.
Una segunda diferencia notable consiste en la posibilidad que tiene el Fortran de definir para los datos de entrada y de salida un **formato**.
En las funciones de impresión del Basic no existen muchos grados de libertad; algunos se han introducido con la opción PRINT USING..., que dentro de ciertos límites permite utilizar formatos de impresión definidos por el usuario. Para las instrucciones de entrada, las limitaciones son también muy notables.
El lenguaje acepta sólo valores homogéneos con las variables, sin posibilidad de definir formatos. En realidad, en los microordenadores y ordenadores personales no existe ningún motivo para disponer de formatos de entrada. La introducción se produce exclusivamente a través de teclado y el operador no obtiene ningún beneficio de poder disponer de formatos particulares. Y viceversa, el Fortran prevé la introducción de los datos también de otros periféricos, como por ejemplo el lector de fichas perforadas. En este caso es útil poder disponer de instrucciones particulares de lectura que permitan reconocer un campo en función de su posición. Esta elasticidad permite perforar las fichas dejando espacios vacíos entre un valor y otro, con un resultado seguramente más legible.
En Fortran, cada instrucción de I/O debe completarse con dos parámetros: el código de la unidad física de I/O y un número de líneas que

especifique qué formato se quiere usar en la representación o en la adquisición del dato.
El periférico debe especificarse, puesto que el Fortran no dispone de instrucciones diferentes para el vídeo y para la impresora. El formato, es decir la forma de representación o de lectura de los datos, puede omitirse y, en este caso, la instrucción se llama **no formateada**.
Las instrucciones de I/O previstas por el Fortran son las siguientes:

READ	para la lectura
PRINT y WRITE	para la escritura

Las palabras READ y WRITE se utilizan también para las funciones I/O en disco, pero este aspecto se ilustrará por separado en la parte dedicada a la gestión de los files.
La forma general de las instrucciones READ y WRITE es la siguiente

READ(N,M) relación de variables
WRITE(K,L) relación de variables

N y K indican los periféricos, respectivamente de entrada y de salida; M y L son los números de línea en que se ha indicado el formato de lectura o de escritura.
La instrucción

READ(5,270) A,B,C

significa: «lee en el periférico 5 según el formato especificado en la línea 270 y transfiere los valores a las variables A,B,C».
Análogamente, la

WRITE(6,300) A,B,C

significa: «escribe los valores de las variables A,B,C en el periférico 6 según el formato especificado en la línea 300».
La instrucción PRINT es una extensión propia de la versión Fortran ANSI 77 que no entra en la forma generalizada del lenguaje. Mientras las anteriores (READ, WRITE) son comunes a todas las versiones, la PRINT no está prevista, por ejemplo, en sistemas mayores que utilizan todavía Compiladores anteriores.
Obsérvese que el número asignado a los periféricos depende de la máquina particular; normalmente, el periférico de entrada es el 5 (teclado o lector de fichas), mientras que el de salida (impresora) es el 6, pero nada impide establecer asignaciones diferentes.

A continuación se expondrán las instrucciones y las funciones de I/O de los microordenadores y ordenadores personales. Por tanto, se dejarán de lado las opciones para el lector de fichas, que es una unidad propia de los grandes sistemas; sin embargo, la extensión es inmediata: basta con variar el código de la unidad.

PRINT

La instrucción de salida direccionada al periférico estándar (vídeo o impresora) es la PRINT. La sintaxis es la siguiente:

PRINT N, datos

N	es el número de línea del formato de salida;
datos	son los valores a escribir y pueden estar constituidos por variables, constantes, cálculos o cadenas.

La indicación del número de línea del formato puede omitirse y, en este caso, los valores relacionados se escribirán siguiendo algunas indicaciones contenidas en la misma instrucción. Examinemos algunos ejemplos.

PRINT 10,A,B,A + B	Escribe, según el formato especificado en la línea 10, las variables A,B y su suma
PRINT *, 'Suma =', A + B	No tiene formato (símbolo *). La salida está sustituida por la cadena «suma =» seguida del valor A + B
ASSIGN 10 TO F PRINT F,A,B,A + B	Es una forma diferente del primer ejemplo. A la línea 10, donde hay especificado el formato, se asigna el nombre simbólico F (ASSIGN 10 TO F) y con este nombre puede utilizarse en cualquier punto del programa

WRITE

La instrucción WRITE puede utilizarse para la transferencia de datos tanto a un periférico general de salida como al disco.
La forma más general es

WRITE (unidad, formato, estado, etiqueta, record) datos

donde

unidad	especifica la unidad de salida
formato	es el número de línea que describe el formato. En algunos casos puede ser una especificación de formato contenida entre comillas
estado	es un código de error; si es diferente de cero indicará el tipo de error producido durante la escritura
label	es el número de línea a la que se transfiere el control ante un error
record	es el número del record para la escritura en files de acceso directo
datos	es una lista de variables, constantes, etcétera, a escribir.

La sintaxis prevé que cada parámetro vaya precedido de una palabra clave que lo identifica:

UNIT	identifica la unidad de salida: UNIT = 6 selecciona la unidad 6
FMT	especifica el formato: FMT = 100 indica el formato descrito en la línea 100
IOSTAT	asigna la variable de estado: si IOSTAT = K, al retorno el código de un eventual error está en la variable K
ERR	indica la línea a que pasar el control en caso de error; la ERR = 500 transferiría el control a la línea 500
REC	indica el número del record: si REC = 5, la instrucción escribe en el record 5 del file (UNIT) el número 6.

Las primeras dos palabras clave (UNIT y FMT) pueden omitirse; las otras, si existen los correspondientes parámetros, son obligatorias. Consideremos los siguientes ejemplos:

WRITE (3,10) A,B	Escribe en unidad 3 según el formato 10; no hay otros parámetros, por tanto no aparecen palabras clave.
WRITE (3,10,ERR=100)	Escribe en la 3 en el formato 10; ante un error, el control pasa a la línea 100.
WRITE (3,10,IOSTAT=I,ERR=100,REC=5)A	Escribe en la unidad 3 en el formato 10; la variable I contiene el eventual código de

El perforador de cinta Facit mod 4070 puede transferir hasta 75 caracteres por segundo.



Facit

error, y ante un error, el control pasa a la línea 100; los datos (en la variable A) están en el record 5.

READ

La instrucción READ tiene dos sintaxis diferentes, según que esté direccionada a la unidad estándar de entrada o hacia un file. En el primer caso, la sintaxis es

READ M, lista

donde M es el número de la línea que contiene las especificaciones de formato; «lista» es una lista de variables en la que se memorizarán los valores leídos. Por ejemplo:

READ 10,A,B	lee las variables A y B según el formato especificado en la línea 10
READ *,(V(I),I=1,5)	lee, con un DO implícito, la variable dimensionada V(I), sin formato
READ 20,(V(I),I=1,5)	la misma situación (DO implícito) con el formato de lectura especificado en la línea 20

El acceso a los files con la instrucción READ requiere el uso de parámetros análogos a los de la WRITE. La forma general es:

READ (unidad, formato, estado, label, record) datos

con los significados vistos anteriormente.
La instrucción

READ (2,1,IOSTAT=KR,ERR=2,REC=6) A,B

lee del record 6 del file 12, según el formato especificado en la línea 1, las variables A y B. El código de error está en KR, y en caso de error, el control se transfiere a la línea 2.
Con respecto a la WRITE, en esta instrucción se ha previsto un nuevo parámetro que determina un salto en caso de EOF (End Of File) en files secuenciales. El parámetro es END = número de línea.
Por ejemplo, la línea

READ (3,150,END=1000,REC=J) V

lee el record J del file secuencial número 3 según el formato 150; en caso de fin de file, el control pasa a la instrucción 1000 (END = 1000).
La opción END = ... tiene un efecto similar a la EOF del Basic, mientras que la ERR = ... puede asimilarse a la ON ERROR... En Fortran, estas dos funciones tienen una gestión más sencilla, puesto que todas las operaciones correlacionadas las realiza el sistema. En Basic debe comprobarse EOF para saber si el file se ha terminado y, por tanto, puede decidirse con un IF si proseguir o no la lectura; en Fortran, todo lo realiza la instrucción READ con la opción END = ...

Formatos de I/O

Todas las funciones de I/O del Fortran pueden referirse a un número de línea en que están definidas las especificaciones según las cuales deben leerse o escribirse los datos. Esta línea particular contiene el formato de adquisición o de presentación y está constituida por la palabra FORMAT seguida de los códigos que expresan, normalmente, el formato. Este tipo de instrucciones normalmente no verifica el Compilador, y los eventuales errores de formato sólo se detectan en la fase de ejecución del programa.
Los principales descriptores de formato, según el tipo de variable, pueden relacionarse así

Enteros	I	
Reales	F	en coma fija (fixed-point)
	E,D	en coma flotante (floating-point)
	G	en coma fija y flotante
Caracteres	A,R	
Lógicos	L	
Octales	K,O	

Formatos para los enteros. La sigla de reconocimiento (I) va seguida del número de cifras que componen el valor a leer o a escribir; I3 indica por ejemplo un valor entero de tres cifras. El código puede ir precedido de otro valor numérico que tiene el significado de **factor de repetición**, indicando cuántas veces debe aplicarse el formato especificado. Por ejemplo, 2I3 indica que el formato I3 debe aplicarse dos veces.
Veamos algunos ejemplos:

READ (6,10) A,B	Lectura de dos valores en el formato 10
WRITE (5,20) I,K,L	Escritura de tres valores en el formato 20

10 FORMAT (I2,I5)	Puede ser un error, puesto que A y B no son enteros por omisión
20 FORMAT (3I2)	En este caso, el formato entero es válido; las variables I,K,L son enteras, sólo debe verificarse que cada una sea de dos cifras (3I2 = tres veces I2)

El formato puede estar incluido en la misma línea que expresa la función; la línea

WRITE (5,'(3I2)') I,K,L

tiene el mismo significado del ejemplo anterior. En algunas formas de Fortran se ha previsto la extensión

In.m

donde

n indica el número de cifras máximo
m el mínimo (en las instrucciones OUTPUT; en la entrada, m se ignora).
Por ejemplo, con el formato I5.3, una variable de valor 4 se escribe de la siguiente manera

ƀƀ004

es decir, dos espacios para llenar el campo máximo previsto (5 de longitud) y 3 caracteres en presentación. Como el valor numérico sólo tiene una cifra (4), las que faltan al campo mínimo (3 de largo) se sustituyen por la cifra 0.

Formatos para los reales. El indicador F debe ir seguido de dos valores numéricos, el primero de los cuales debe indicar la longitud total del campo en caracteres (comprendido el punto decimal) y el segundo, el número de cifras después de la coma. En la primera tabla de abajo se han representado algunos ejemplos de especificaciones de este formato.
Los indicadores E y D se utilizan para representaciones en coma flotante: el código E reserva al exponente dos posiciones y el código D tres posiciones. En la segunda tabla de abajo pueden verse algunos ejemplos. La longitud del campo (10 en el segundo ejemplo) comprende tres posiciones para el exponente (opción D) y una para el punto decimal; por tanto, un campo de 7 de largo en formato D tiene sólo tres posiciones útiles para el valor.
El formato E funciona exactamente como el formato D; la única diferencia consiste en el exponente, al que sólo se le reservan dos cifras.
El formato G puede utilizarse tanto para la representación en coma fija como para la coma flotante puesto que se adecua a la longitud del valor a representar.

ESPECIFICACIONES DE FORMATO PARA NUMEROS REALES EN COMA FIJA (FIXED-POINT)

Valor en memoria	Descriptor	Salida
9.7584	F 4.2	9.76
9.7584	F 6.4	9.7584
15	F 4.1	15.0
128.2	F 3.1	Error: el valor no entra en el campo

ESPECIFICACIONES DE FORMATO PARA NUMEROS REALES EN COMA FLOTANTE (FLOATING-POINT)

Valor en memoria	Descriptor	Salida
21.412	D 7.4	Error: el campo de longitud 7 no puede contener el dato
21.412	D 10.4	2.1412D + 01

Formatos para los caracteres. El descriptor de formato de caracteres para I/O es la letra A seguida del número de caracteres previstos; A3 indica una cadena de tres caracteres.
Si los caracteres que componen la cadena son más del número previsto en la descripción, sólo se consideran los caracteres especificados en el formato, a partir de la izquierda.
Por ejemplo, dada la cadena ABCDEFG, en la salida el formato A4 genera la impresión ABCD; el mismo formato utilizado en lectura sólo transfiere los primeros cuatro caracteres.
El formato R (extensión del Fortran ANSI 77) es análogo al anterior pero acerca los valores a la derecha en el campo de I/O.

Formatos para las constantes lógicas. El formato para una constante lógica (TRUE/FALSE) es indicado por la letra L seguida de un número que representa la longitud del campo a examinar. Este campo debe contener como primer carácter (diferente de espacio) la letra T (TRUE) o F (FALSE).
Por ejemplo, en la entrada, el formato L3 examina un campo de tres caracteres y asigna a la variable indicada el valor TRUE si encuentra la letra T y FALSE si encuentra la letra F. Introduciendo una de las cadenas

T, Tx, Txy

el resultado en la variable es T (TRUE); las letras que siguen al código T no se consideran.
Análogamente, las cadenas F, FZ, FKL transfieren en la variable el valor F (FALSE).
En la salida se tiene la inversión de la letra T o F precedida por tantos espacios como sean necesarios para completar el campo especificado en el formato.
Por ejemplo, si la variable contiene TRUE, el formato L4 origina ƀ ƀ ƀ T, mientras que el formato L1 proporciona T.

Descriptores de edit

La gestión completa de las funciones de I/O requiere el uso de descriptores particulares, llamados **descriptores de edit**, los cuales se utilizan para insertar comentarios, espacios o ceros en los campos de los datos y se dividen en tres categorías:

- descriptores para funciones de entrada
- descriptores para funciones de salida
- descriptores para funciones de entrada y de salida.

Edit en entrada. Los descriptores son BN y BZ. Con los descriptores numéricos ya descritos se ignoran los eventuales espacios en blanco entre los valores numéricos introducidos; en cambio, con el descriptor BZ se convierten en ceros.

Edit en salida. Las principales funciones de estos descriptores son dos:

- impresión de cadenas, comentarios y descripciones
- presentación del signo algebraico

La impresión de cadena puede obtenerse de dos maneras. Una primera posibilidad consiste en indicar los caracteres a escribir encerrados entre comillas, o bien los mismos caracteres pueden describirse como constantes Hollerith (nH). Por ejemplo, los formatos

14,'Total =',I3
14,9HTotal =,I3

producen la impresión de un valor numérico (I4), seguido de la leyenda Total = y de otro valor numérico (I3).
Normalmente, en la impresión de valores numéricos positivos, el signo + no se presenta, al contrario de lo que sucede para los valores negativos, para los cuales el signo – siempre se indica. El descriptor SP imprime también del signo +. Como BZ, también SP permanece activo mientras no encuentra el descriptor SS o el fin de la especificación de formato.

Descriptores comunes. Los principales descriptores utilizados tanto en entrada como en salida son los siguientes:

nX:salta n posiciones
Tn:tabula en la columna n
/:indica el final de un record

Factor de escala. Se indica con el símbolo nP, donde n se interpreta como factor multiplicativo, mientras que P es el indicador. El factor de escala puede aplicarse sólo a valores numéricos, y por tanto con los formatos F, E, etc., y puede ser tanto positivo como negativo. Por ejemplo, con el formato F10.4, el valor 135.79817 se es-

cribe en la forma 135.7981 (cuatro decimales). Aplicando un factor de escala, por ejemplo – 2, el formato se convierte en: – 2 PF10.4 y la impresión es 1.357981.

Repeticiones de formato. Un formato cualquiera puede repetirse un número cualquiera de veces indicando el número de las repeticiones antes del descriptor. Por ejemplo, la especificación 3F5.2 repite tres veces el formato F5.2. El factor de repetición puede aplicarse también a formatos constituidos por más descriptores, incluyendo estos últimos entre paréntesis. Por ejemplo, la especificación

(3I2,2(I4,X,F8.2))

describe la siguiente salida:

3I2 tres valores enteros de 2 cifras
2 () repetición por dos veces de un grupo constituido por:
- I4 un valor entero de 4 cifras
- X un espacio
- F8.2 un valor real con un campo de 8 caracteres y dos decimales.

Obsérvese que los tres valores enteros (3I2) se escribirían consecutivamente, y resultarían difíciles de distinguir. Análogamente, los dos grupos I4,X,F8.2 se escribirían seguidos.
Para separar los diferentes campos puede utilizarse el descriptor X.

Por ejemplo, el formato anterior, escrito en la forma

(3(X,12),2(X,14,3X,F8.2))

inserta un espacio entre las primeras tres variables enteras (X,I2) y un espacio entre los grupos I4, etc.
En el gráfico de esta página se ha representado una comparación entre las dos salidas.

Instrucciones de I/O no formateadas

Las instrucciones de I/O sin formato tienen una sintaxis y una lógica similares a las correspondientes instrucciones Basic para el vídeo, para la impresora y para el teclado, mientras que las instrucciones para la gestión del disco difieren notablemente.

Instrucciones de entrada. La instrucción de lectura es la misma que la descrita al hablar de las instrucciones con formato, con el símbolo * además:

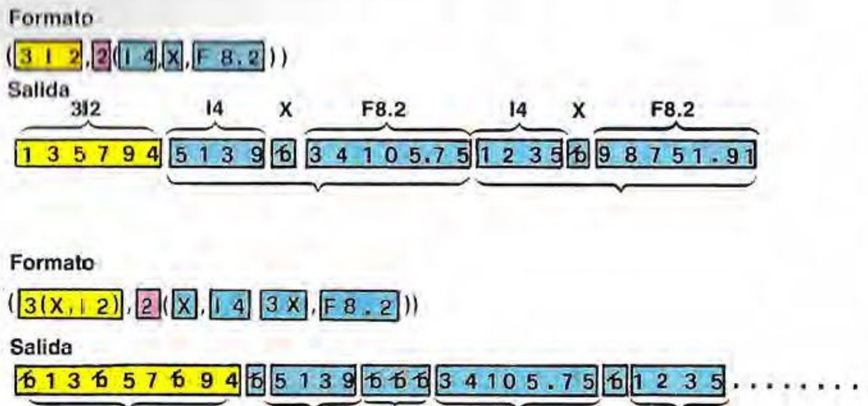
READ *, variables

o bien

READ (U, *) variables

En la primera, el uso de la unidad de sistema (teclado) es implícito. La segunda puede direccionar una unidad de entrada cualquiera con el código numérico correspondiente (U).

EJEMPLOS DE FORMATOS Y SUS RESPECTIVAS SALIDAS



Los valores en lectura, que se memorizarán en las respectivas variables, deben estar separadas por uno o más espacios, por el símbolo / (slash) o por una coma. Por ejemplo, dada la instrucción

READ *, A, B, C

los valores A = 3, B = 5, C = 11 pueden introducirse con una de las formas

3 5 11
3 / 5 / 11
3,5,11.

Si se quiere asignar a todas las variables el mismo valor numérico puede utilizarse un factor de repetición. Por ejemplo, para asignar a las variables A,B,C el valor 7, puede utilizarse indiferentemente una de las formas:

7 7 7
7 / 7 / 7
7,7,7
3 * 7 (significa 3 veces el valor 7)

Los eventuales espacios en blanco no se consideran y la correspondiente variable conserva el propio valor, siempre teniendo en cuenta la instrucción READ.
Por ejemplo, si la instrucción de lectura es

READ *,A,B,C

proporcionando a la entrada los valores

ƀ, 20, ƀ

se tiene la transferencia del valor 20 en la variable B, mientras que A y C permanecen sin alteración (el valor que se tenía antes de la ejecución de la instrucción READ).

Instrucciones de salida. Las dos formas de la instrucción PRINT no formateadas son

PRINT *, variables
PRINT (U, *) variables

con los mismos significados antes vistos.
Los valores se escriben de manera diferente según el tipo de las variables:

Enteros	escritos como números enteros
Reales	pueden tener o no el exponente, en función del valor
Doble precisión	como los reales
Complejos	dos valores (parte real y parte imaginaria)
Lógicos	se imprime la letra T si el valor es TRUE y la letra F si es FALSE.

Consideremos el siguiente ejemplo: dadas algunas variables a las que se les han asignado los valores

I = 256	entero
R = 25.72	real
D = 0.1173D2	doble precisión
C = (12,5)	complejo
L = .TRUE.	lógico
CH = 'TEXTO'	caracteres

Algunas salidas podrían ser las siguientes:

Instrucciones	**Salida**
PRINT *,I,CH	256 TEXTO
PRINT *,C,L	(12.,5.) T
PRINT *,R,D	25.72 1.173D1

Ejemplo de programación en Fortran

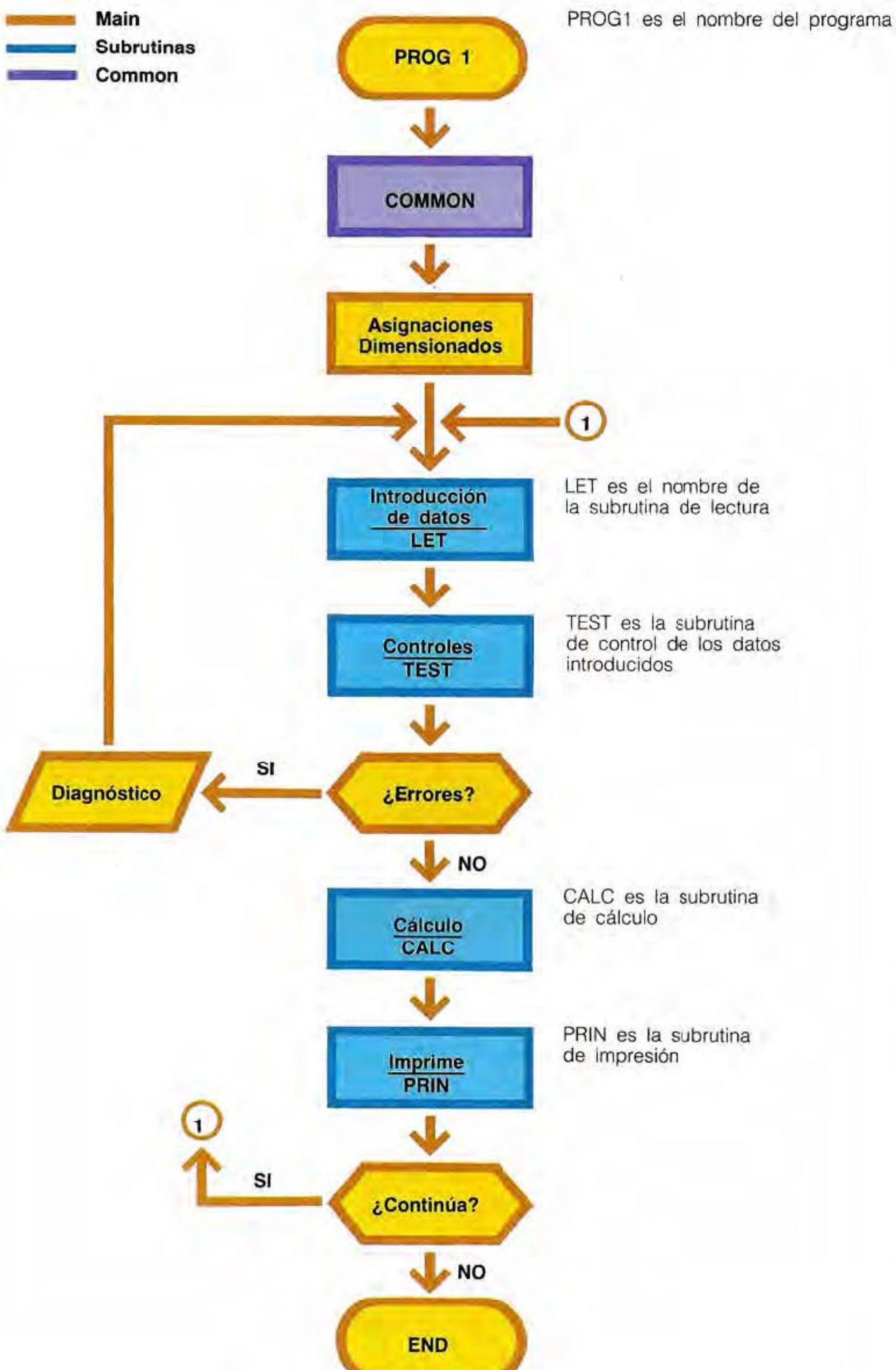
En el gráfico de la página siguiente se indica el diagrama de flujo de un programa que muestra el uso de algunas instrucciones Fortran.
El problema a resolver es el siguiente.
Dada una cifra inicial invertida con capitalización mensual, calcular el interés al final de cada mes. La tasa puede variar mes a mes y, por tanto, deben preverse 12 valores (uno por mes). El cálculo se realiza como sigue:

1 / Al final de cada período (mes) el interés viene dado por:

$$\text{Interés} = \frac{\text{Capital} \times \text{Tasa (del mes)}}{100}$$

2 / El capital del mes siguiente viene dado por:
Capital mes siguiente =
= Interés a fin de mes (punto 1) + Capital anterior.

DIAGRAMA DE FLUJO PARA EL CALCULO DE INTERESES



Indicando
C(12) = Capital principio mes (matriz)
S(12) = Interés devengado en el mes (matriz)
T(12) = Tasa a aplicar cada mes (matriz)
Para un mes genérico I se tiene en la fórmula:

S(I) = (C(I) * T(I))/100 (Interés a fin de mes)
C(I + 1) = S(I) + C(I) (Capital mes siguiente)

Para realizar el cálculo en los 12 meses basta con aplicar las dos fórmulas en un bucle con I variables entre 1 (primer mes) y 11; el último valor (mes 12) se calcula por separado, puesto que representa el valor final (total anual).

En los gráficos de las páginas 1185, 1187, 1188 y 1190 se han representado los diagramas de flujo de las subrutinas utilizadas. Los listados en las dos versiones Basic y Fortran pueden verse en las páginas 1184, 1186, 1189, y 1191. En el main de abajo se presentan algunas de las principales diferencias que existen entre los dos lenguajes. En particular, las líneas 10 y 20 del Basic son comentarios, mientras que para la versión Fortran constituyen instrucciones indispensables. La primera línea (correspondiente a la 10) define el nombre del programa y la segunda establece algunas variables de uso común a las rutinas.

PROGRAMA PRINCIPAL

Versión Basic

```
10  REM : PROG1
20  REM : no sirve common
25  DIM C(12),S(12),T(12)
30  GOSUB 1000
40  GOSUB 2000
50  IF KE=0 THEN GOTO 80
60  PRINT "ERROR"
70  GOTO 30
80  GOSUB 3000
90  GOSUB 4000
100 PRINT "CONTINUA 2"
110 INPUT N
120 IF N=1 THEN GOTO 30
130 :
140 REM========================
150 REM *FALTAN LOS FORMATOS*
160 REM========================
170 :
180 END
```

Versión Fortran

```
      PROGRAM PROG1
      COMMON C(12),S(12),T(12)
30    CALL LET
      CALL TEST (KE)
      IF (KE.EQ.0) GOTO 80
      WRITE (6,900)
      GOTO 30
80    CALL CALC(TA)
      CALL PRIN(TA)
      PRINT 910
      READ (6,920)N
      IF (N.EQ.1) GOTO 30
C * * COMENTARIO * *
800   FORMAT(3X,6HError)
910   FORMAT(3X,'Continua 2')
920   FORMAT(X,11)
      STOP
      END
```

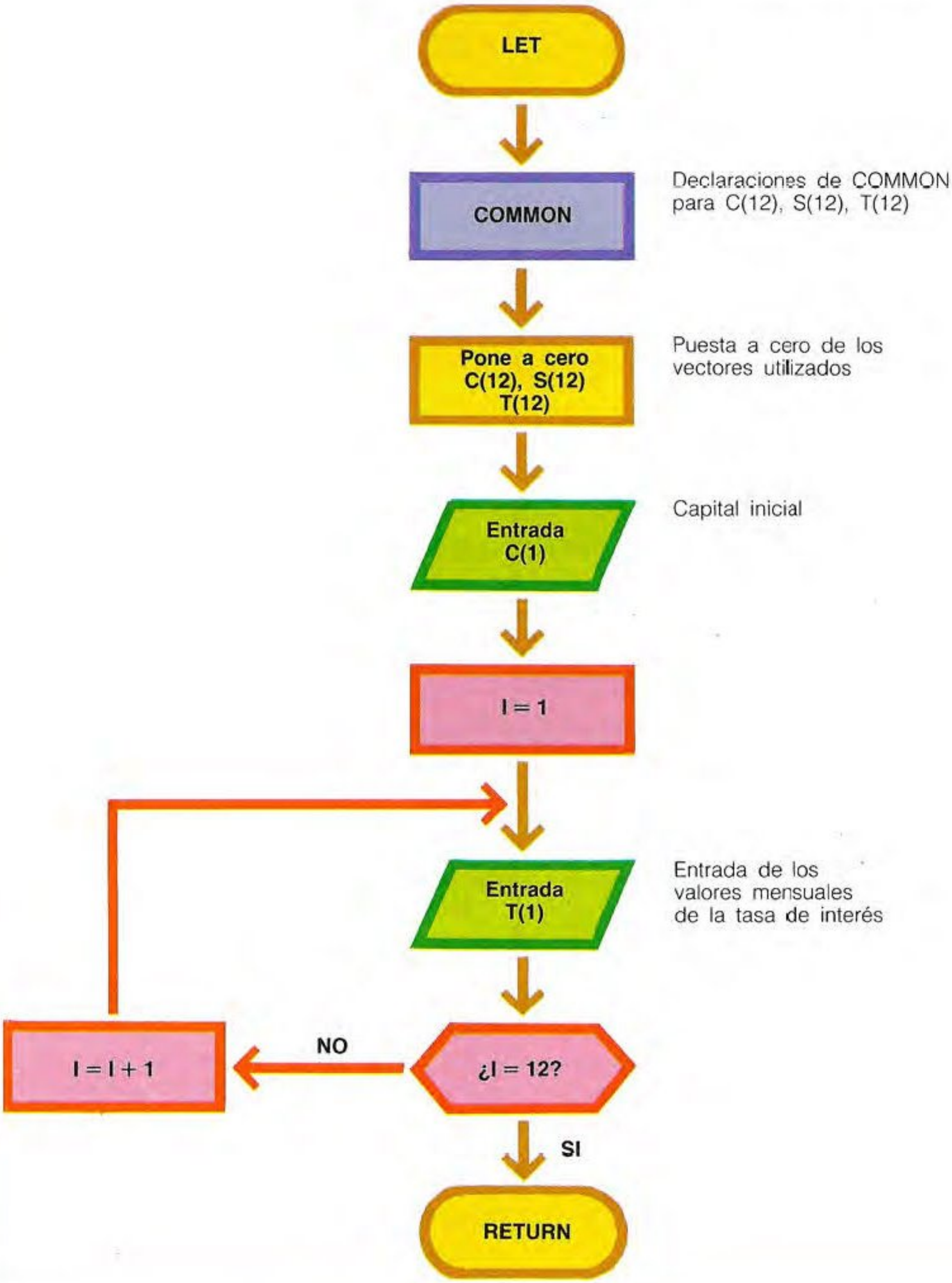
SUBRUTINA DE INTRODUCCION DE DATOS

- **Common**
- **Funciones I/O**
- **Bucle**

Entradas Ninguna

Funciones Puesta a cero de las variables
Lectura de capital inicial
Lectura de la tasa de interés
en doce meses

Salidas Capital inicial [C(1), inicio = mes 1]
Tasas de interés [T(12), 12 valores]



SUBRUTINA DE INTRODUCCION DE DATOS

Versión Basic

```
1000 REM
1010 REM
1020 FOR I=1 TO 12
1030 C(I)=0:S(I)=0:T(I)=0
1040 NEXT I
1050 INPUT "Capital ";C(1)
1060 FOR I=1 TO 12
1070 PRINT "Mes=";I
1080 INPUT ."Tasa";T(I)
1090 NEXT I
1100 RETURN
```

Versión Fortran

```
        SUBRUTINA LET
        COMMON C(12),S(12),T(12)
        DO 10 I=1,12
        C(I)=0.
        S(I)=0.
        T(I)=0.
 10     CONTINUE
        READ 90,C(1)
        DO 20 I=1,12
        PRINT 92,I
        READ 94,T(I)
 20     CONTINUE
 90     FORMAT (X,F8.0)
 92     FORMAT (X, 'Mes=',X,12)
 94     FORMAT (X,F5.2)
        RETURN
        END
```

En la versión Fortran sólo se han numerado las líneas que deben direccionarse, en particular las 900, 910 y 920, que describen los formatos de I/O y carece de equivalente en la versión Basic.

En este ejemplo se han adoptado las dos formas posibles de la instrucción de salida: WRITE y PRINT. En la primera se especifica la unidad de salida (la número 6); en la segunda no es necesario, ya que la instrucción asume la unidad estándar de sistema. Por tanto, la segunda forma es muy similar a la análoga en Basic (línea 100), con la diferencia de que en Fortran se hace referencia a un formato (línea 910).

Gestión de los files

El Fortran, del mismo modo que muchos otros lenguajes nacidos en grandes procesadores, ve el disco como otra unidad de entrada o salida cualquiera.

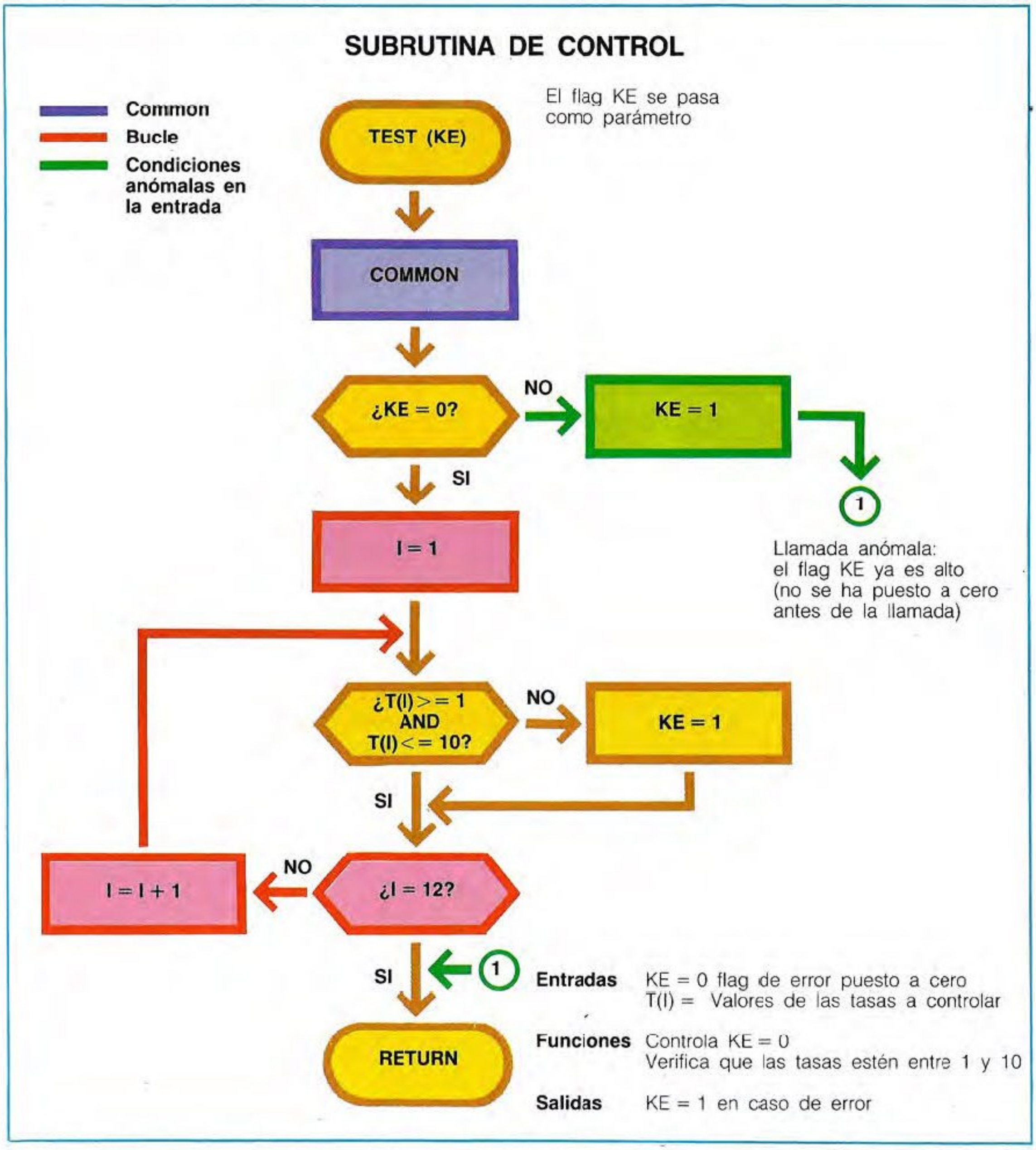
En general, todas las instrucciones de I/O pueden direccionarse hacia una unidad genérica, entre las cuales hay el disco; la selección se realiza especificando el código que identifica la unidad o el file.

Para algunas unidades, el código numérico está definido y anotado a priori en el sistema (por ejemplo, uso de la PRINT hacia la unidad estándar); para otras (files en disco) debe ser el programador el que lo asigne.

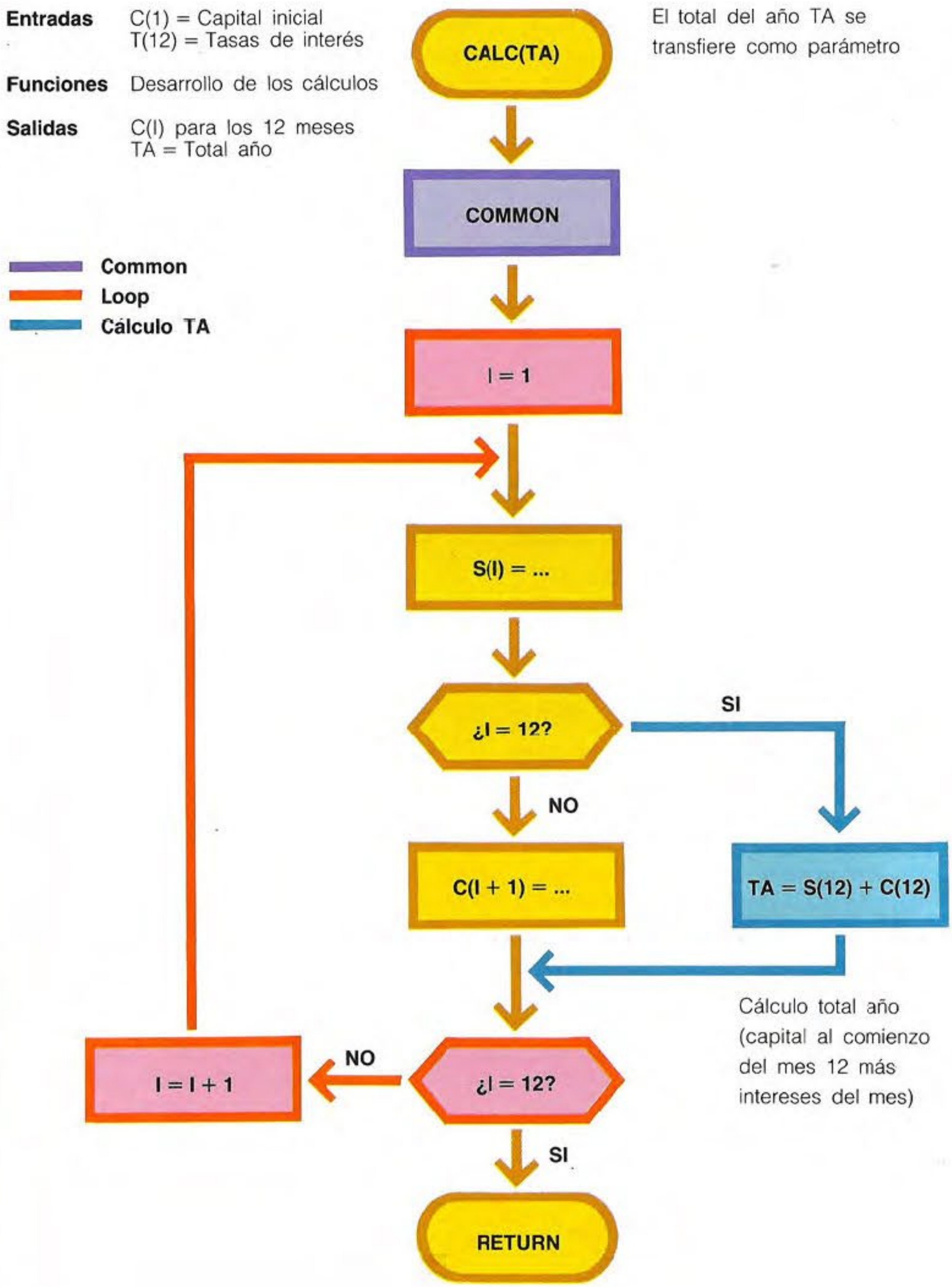
Las unidades del primer tipo se llaman **preconectadas**; las del segundo tipo deben ser "conectadas" con un número de unidad lógica por medio de la instrucción OPEN.

Por tanto, encontramos un concepto diferente de utilización de los periféricos.

En el Basic (estándar), además del disco no existen otros dispositivos que gestionen files. En el Fortran (como también en el Cobol y en otros lenguajes), las otras unidades también pueden ser vistas como files. Así puede utilizarse el file asignado a la unidad de vídeo, a la impresora o al teclado y, en los sistemas más grandes, a la unidad de cinta.
Cada una de estas unidades, si no está asignada por omisión por el sistema, debe conectarse con comandos e instrucciones explícitos.
Además, el Fortran prevé un tipo de file particular, que recibe la denominación de **file interno**, constituido por una zona de memoria a la que puede hacerse referencia como si se tratase de un file «externo» cualquiera, por ejemplo residente en disco.
Cada variable de esta área de memoria o cada elemento de una matriz, siempre que esté definido como parte de un file interno, se considera como un record.
Para realizar las operaciones de I/O debe sustituirse en las respectivas instrucciones el nombre simbólico de la variable o de la matriz por el



SUBRUTINA DE CALCULO



SUBRUTINAS DE CONTROL Y CALCULO

Versión Basic

```
2000 REM
2010 REM
2020 IF KE<>0 THEN GOTO 2080
2030 FOR I=1 TO 12
2040 IF T(I)<1 THEN KE=1
2050 IF T(I)>10 THEN KE=1
2060 NEXT I
2070 RETURN
2080 KE=1
2090 RETURN
3000 REM
3010 REM
3020 FOR I=1 TO 12
3030 S(I)=(C(I)*T(I))/100
3040 IF I=12 THEN GOTO 3060
3050 C(I+1)=S(I)+C(I)
3060 NEXT I
3070 TA=S(12)+C(12)
3080 RETURN
```

Versión Fortran

```
      SUBROUTINE TEST (KE)
      COMMON C(12),S(12),T(12)
      IF(KE.NE.0) GOTO 20
      DO 10 I=1,12
      IF(T(I).LT.1)KE=1
      IF(T(I).GT.10)KE=1
10    CONTINUE
      RETURN
20    KE=1
      RETURN
      END

      SUBROUTINE CALC(TA)
      COMMON C(12),S(12),T(12)
      DO 10 I=1,12
      S(I)=(C(I)*T(I))/100.
      IF I=12 GOTO 10
      C(I+1)=S(I)+C(I)
10    CONTINUE
      TA=S(12)+C(12)
      RETURN
      END
```

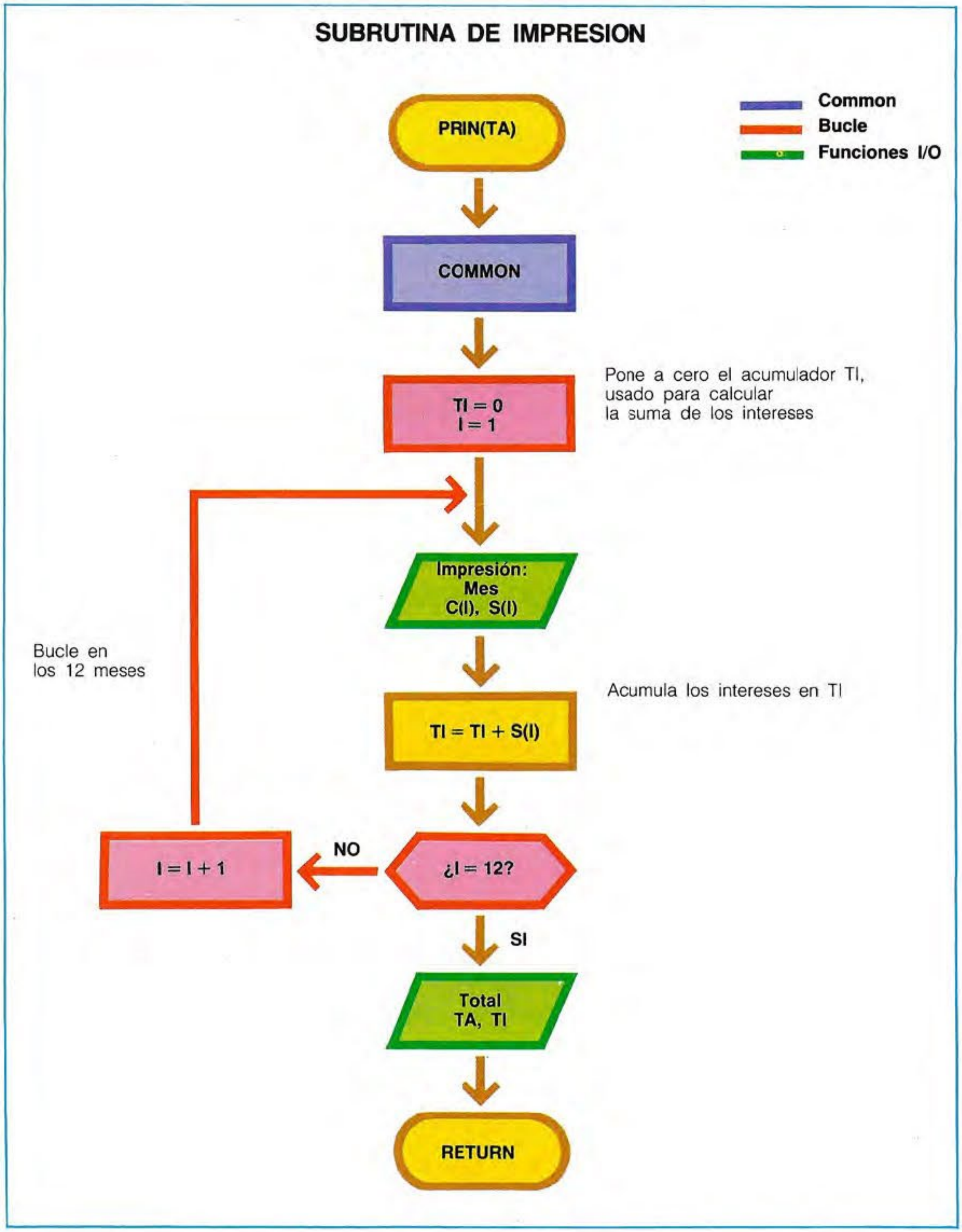
número de unidad lógica (file). Por ejemplo, la instrucción:

WRITE(6,30)...

en su uso normal se refiere a la unidad 6 (con formato 30). Para direccionarla a un file interno se debe sustituir el valor 6 con el nombre de la variable (o de la matriz) considerada como file interno. La línea

WRITE(PRUEBA,30)V

transfiere a la variable PRUEBA el contenido de la variable V según el formato definido en la línea 30.



Gestión de los files en disco

Muchas de las instrucciones previstas para el disco son válidas para todos los demás files. A continuación se hará referencia sólo a la unidad de disco por tratarse de la unidad más importante al hablar de files; para la extensión a las demás unidades deben consultarse los manuales específicos de la máquina empleada.

El Fortran gestiona dos tipos de files: los de acceso secuencial y los de acceso directo, con los

SUBRUTINA DE IMPRESION

Versión Basic

```
4000 REM
4010 REM
4020 TI=0
4030 FOR I=1 TO 12
4040 PRINT "Mes=":I,C(I),S(I)
4050 TI=TI+S(I)
4060 NEXT I
4070 PRINT "Tot. :";TA,TI
4080 RETURN
```

Versión Fortran

```
      SUBROUTINE PRINT(TA)
      COMMON C(12),S(12),T(12)
      TI=0.
      DO 20 I=1,12
      WRITE (6,30) I,C(I),S(I)
      TI=TI+S(I)
 20   CONTINUE
      WRITE (6.*) TA.TI
 30   FORMAT (X, 'Mes:',X,I2,8X,F9.1,X,F8.1)
      RETURN
      END
```

mismos significados que en el Basic. Los tipos de records previstos son tres:

- **Records formateados*** los datos se leen y escriben según precisas especificaciones de formato
- **Records no formateados** no tienen formato y contienen una serie de datos tal como se encuentra en memoria
- **Records End-Of-File** es un record que no contiene datos y sirve para indicar el final de un file secuencial. Se escribe con la instrucción ENDFILE.

* El término «formateado» es una mala españolización de un vocablo inglés ampliamente adoptada.

Las principales instrucciones correlacionadas a los files se mencionan a continuación:

OPEN	conecta (abre) un file
READ	lectura de datos
WRITE	escritura de datos
BACKSPACE	desplaza dentro de un record el puntero de un file secuencial
REWIND	desplaza el puntero de un file secuencial sobre el primer record
ENDFILE	escribe el record End-Of-File, terminador de un file secuencial
INQUIRE	proporciona informaciones relativas a un determinado file
CLOSE	desconecta (cierra) un file

OPEN. La instrucción OPEN sirve para «conectar» el file con el sistema asignándole un número de unidad lógica. Contiene algunas definiciones del file, análogas a las utilizadas en Basic. Los parámetros a proporcionar son:

UNIT número de unidad lógica asociado: poniendo UNIT = 3 el file es reconocido como unidad 3

IOSTAT especifica la variable (entera) en que transferir eventuales códigos de error IOSTAT = KR transfiere a KR el eventual código de error

ERR indica el número de la línea a la que transferir el control si se verifica un error: ERR = 130 significa «GO TO 130 al producirse un error»

FILE especifica el nombre del file: FILE = 'TEST'; si el file no existe, el sistema lo genera en el momento de la ejecución de la OPEN

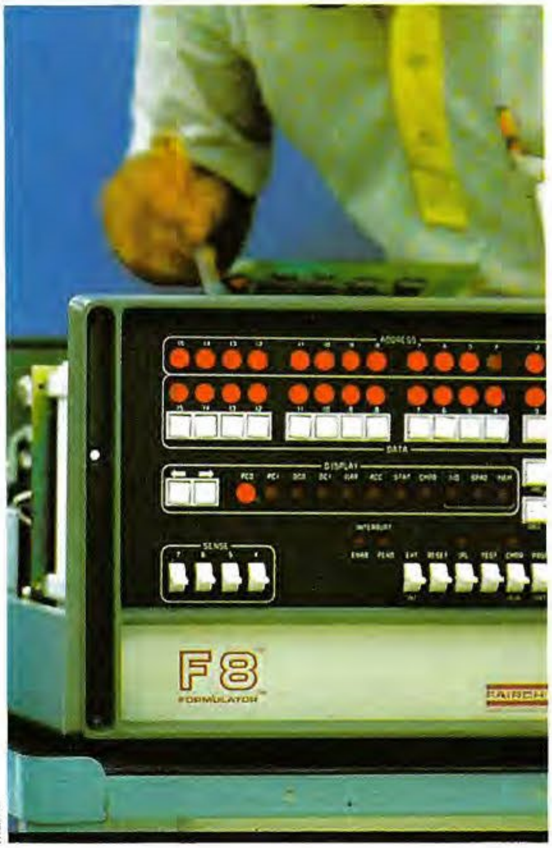
STATUS verifica o no la existencia del file. Los valores admitidos son

STATUS = 'OLD' el file debe existir; en caso contrario genera un error

STATUS = 'NEW' el file no debe existir; si existe genera error

STATUS = 'SCRATCH' no debe proporcionarse ningún nombre; se crea un file de trabajo

Cuadro de monitorizado de Fairchild.



Marka

STATUS = 'UNKNOWN' determina la búsqueda; si el file no existe se crea

Las primeras dos opciones (OLD, NEW) sirven para evitar errores abriendo respectivamente un file que ya debe existir o uno que no debe estar presente; la última (UNKNOWN) gestiona automáticamente la selección y conecta el file especificado si existe, o lo genera (y a continuación lo conecta) si no existe. La opción UNKNOWN se asigna por omisión

ACCESS Método de acceso; puede asumir los valores

DIRECT Acceso directo; en la siguiente I/O debe especificarse el número de record (REC = ...)

SEQUENTIAL File secuencial; es el valor por omisión

FORM Especifica si los datos deben formatearse o no; puede tener los dos valores FORMATTED o UNFORMATTED

RECL Longitud del record en bytes para acceso directo

BLANK Puede asumir los valores NULL o ZERO; en el primer caso se ignoran los eventuales espacios en blanco entre los valores numéricos; en el segundo se convierten en ceros. Por ejemplo, 15ƀ79 con la opción NULL se convierte en 1579, con la opción ZERO 15079. Por omisión se asume NULL.

Un ejemplo de uso de las especificaciones asociadas a la OPEN puede ser el siguiente

```
OPEN (2,IOSTAT = K,ERR = 1,FILE = 'F1')
```

que define el file F1 como secuencial (por omisión) conectado a la unidad lógica 2; K es la variable de error y 1 el número de línea para la gestión de las condiciones de error.

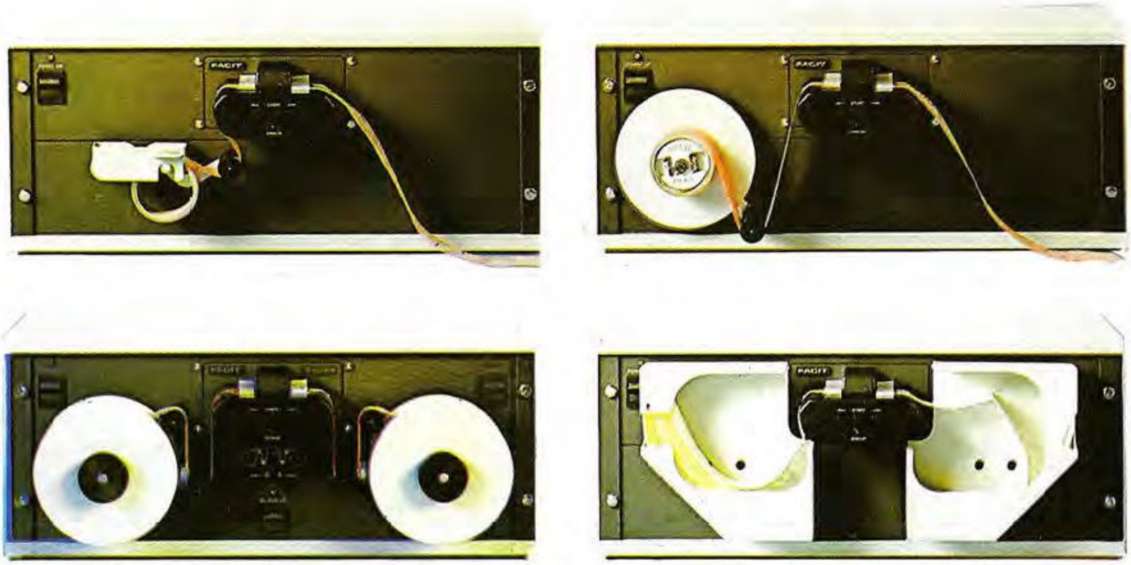
La sintaxis completa para definir la unidad lógica es UNIT = 2, pero UNIT puede omitirse.

La línea

```
OPEN (2,FILE = 'F1',ACCESS = 'DIRECT',
RECL = 1)
```

abre el file F1 de acceso directo, con records de un byte de longitud, en la unidad lógica 2.

La línea



Facit

Diversas configuraciones del lector de cinta perforada Facit 4030. Arriba a la izquierda: configuración para la lectura de cintas que contienen menos de 8000 caracteres (unos 20 m de longitud); a la derecha: configuración para la lectura de cintas que contienen hasta 2000 caracteres (unos 50 m). Arriba a la izquierda: configuración para la lectura de cintas que contienen hasta 40.000 caracteres (100 m); a la derecha: configuración para la lectura de cinta plegada.

```
OPEN (2,FILE = 'TEST',STATUS = 'OLD',
ACCESS = 'DIRECT',RECL = 100)
```

realiza las mismas funciones de la instrucción anterior, pero controla que el file ya exista (STATUS = 'OLD'); si no es así, genera un error. En cambio, la otra forma, en caso de error habría creado el file, puesto que por omisión es STATUS = 'UNKNOWN'.

READ y WRITE. Las instrucciones READ y WRITE aplicadas a los files en discos tienen formatos idénticos a las análogas utilizadas para otros periféricos. Una vez abierto un file, después de haberle asignado un número de unidad lógica, las operaciones de I/O son idénticas a las utilizadas para los otros periféricos.
Naturalmente, para los files directos debe especificarse el número del record a utilizar. Por ejemplo, la línea

```
READ (2,130,REC = 12)variable1,variable2,...
```

lee según el formato 130 el record número 12 del file número 2 y transfiere los valores a las variables especificadas. Análogamente en escritura, con la línea

```
WRITE (2,130,REC = 7) variables
```

los valores de las variables se transfieren al record número 7.

BACKSPACE. En los files secuenciales después de cada operación de I/O se incrementa un puntero que indica el número del primer record disponible. En la siguiente operación procesa el record siguiente, y así sucesivamente. La instrucción BACKSPACE permite decrementar en uno este puntero, posicionándolo nuevamente sobre el último record procesado.
La sintaxis de la instrucción es la siguiente

```
BACKSPACE n
```

con n número del file.

REWIND. Posiciona el puntero al principio del file. Los parámetros son UNIT, IOSTAT, ERR, cada uno con los propios significados que tienen en la OPEN. Por ejemplo, la línea

```
REWIND(2,IOSTAT = K,ERR = 100)
```

posiciona el puntero al principio del file 2; la variable de error es K y, en caso de error, el control se transfiere a la instrucción 100.

ENDFILE. Escribe un record de fin de file; el

único parámetro es el número de unidad. La línea

ENDFILE 2

termina el file abierto anteriormente con el número 2. Las dos instrucciones REWIND y ENDFILE introducidas en este orden producen la pérdida del contenido del file, puesto que la primera posiciona el puntero al principio y la segunda marca esta posición como End-Of-File (EOF). El resultado es un file con un solo record que contiene el final del file.

INQUIRE. Proporciona las características de un file. Los parámetros son los mismos que los utilizados en la OPEN, con valores que dependen del estado del file. Por ejemplo, la instrucción

INQUIRE (FILE = 'PRUEBA',OPENED = K,
NUMBER = N,ACCESS = A)

restituye

K = TRUE si el file es abierto (si no, FALSE)
N = Número de unidad lógica asociada
A = Método de acceso

CLOSE. Cierra el file especificado y lo disocia del número de unidad lógica. Los parámetros previstos son

UNIT	Número de la unidad
IOSTAT	Estado de error
ERR	Número de línea para la gestión de los errores
STATUS	Puede asumir dos valores: KEEP o DELETE. En el primer caso, el file sólo se cierra, en el segundo se anula. Por omisión se asume KEEP.

En el gráfico de la página siguiente se ha representado el diagrama de flujo de un programa que lee un file secuencial y calcula la media de los valores (numéricos) contenidos en él.
Cada valor (variable V) ocupa un record y está constituido por un número real de tres cifras enteras y una cifra decimal (formato F5.1).
En el diagrama de flujo se ha representado el control del fin de file (que no aparece en el listado de la página 1196 porque ya está implícitamente incluido en la instrucción READ, línea 100) con la opción END = 200.

El file a abrir se llama PRUEBA y está asociado a la unidad lógica 1. Obsérvese la línea 200. Para el cálculo de la media de los valores no puede utilizarse el nombre MEDIA porque éste empieza con una de las letras que definen variables enteras (I, J, K, L, M, N); utilizándolo se tendría la pérdida de eventuales valores decimales. Como se ve en el listado, los formatos pueden encontrarse en un punto cualquiera del programa, pero antes de la instrucción END.
Esta instrucción debe ser la última porque constituye una señal para el Compilador; las eventuales líneas presentes después de la END no se procesan y, por tanto, quedan excluidas del programa. El listado muestra la notable sencillez de las operaciones de I/O en el disco, propias del Fortran. Las variables pueden leerse o escribirse en un formato cualquiera, y no son necesarias ulteriores instrucciones. En Basic habría sido necesario transformar por programa los valores, leídos como caracteres en números, dado que las operaciones I/O hacia el disco sólo gestionan caracteres.

Instrucciones particulares del Fortran

A este grupo pertenecen instrucciones propias de las formas de programación más avanzadas. Se mencionan sólo a título indicativo, puesto que su existencia depende del tipo de máquina y de Compilador utilizados.
Las principales son:

EXTERNAL
ENTRY
SAVE
BLOCK DATA

EXTERNAL

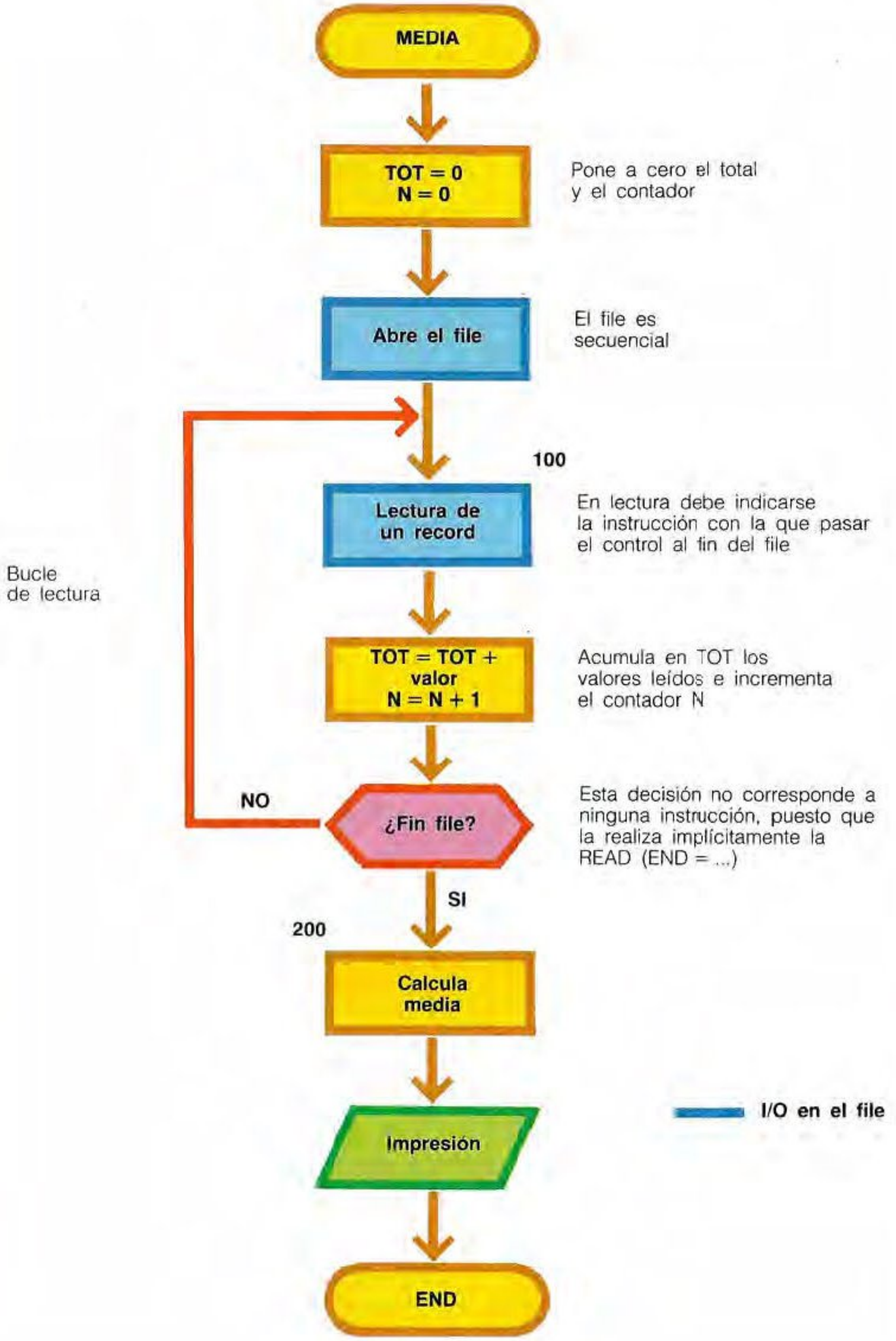
Define un nombre de subrutina y permite su uso como argumento en una llamada. La sintaxis es

EXTERNAL nombre 1, nombre 2, etc.

donde nombre 1, nombre 2,... son los nombres de las subrutinas a transferir. Por ejemplo, las instrucciones

```
EXTERNAL A
CALL SUMA (A,B,C)
SUBROUTINE SUMA (A,B,C)
X = A(N) + B + C
RETURN
```

CALCULO DEL VALOR MEDIO DE LOS DATOS CONTENIDOS EN UN FILE



LECTURA DE UN FILE Y CALCULO DE LA MEDIA DE LOS VALORES

```
*PROGRAM MEDIA
TOT=0,
N=0
       OPEN(1,FILE= 'PRUEBA',ACCESS='SEQUENTIAL')
  100  READ (1,110,END=200) V
       TOT=TOT+V
       N=N+1
       GOTO 100
  200  RMEDIA=TOT/N
       WRITE (6,210) RMEDIA
       CLOSE (1)
       STOP
  110  FORMAT(F5.1)
  210  FORMAT (X,'MEDIA=';X,F9,1)
       END
```

informan al Compilador que el nombre simbólico A no indica una variable, sino otra subrutina (o función) que a su vez podrá tener argumentos en una llamada [SUMA(A,B,C)]. Donde el sistema encuentra el nombre A utiliza la subrutina (o función) correspondiente.
Por ejemplo, la línea

X = A(N) + B + C

utiliza A como función del parámetro N, mientras que B y C son variables. La declaración EXTERNAL también puede utilizarse para definir las funciones de usuario que tienen el mismo nombre que las de sistema. Por ejemplo,

EXTERNAL SQRT

define una función de nombre SQRT que realiza cálculos definidos por el programador, y no la raíz cuadrada como realizaría si el nombre se refiriese a la función de sistema. Evidentemente, esta última ya no está disponible.

ENTRY

Define un punto de entrada alternativo en una subrutina. La sintaxis es:

ENTRY nombre (argumentos)

nombre — es el nombre simbólico de esta nueva entrada (entry-point)

argumentos son las magnitudes a intercambiar.

Por ejemplo, en la subrutina

SUBROUTINE PRUEBA(A,B,C)

....

(desarrollo cálculos)

....

ENTRY PRUEBA1 (C)

....

RETURN

se ha previsto un ingreso normal (SUBROUTINE...) con los tres parámetros A,B,C, y una entrada alternativa (ENTRY...) de nombre PRUEBA1 con el solo parámetro C.
En llamada, una CALL PRUEBA(A,B,C) activa la subrutina desde su inicio, mientras que la CALL PRUEBA1 (C) transfiere el control a la línea ENTRY... saltando las líneas anteriores comprendidas entre SUBROUTINE... y ENTRY...
En el gráfico de al lado se ha esquematizado la diferencia entre los dos tipos de llamada.

SAVE

Permite conservar los valores de algunas variables después del retorno de una subrutina. Por ejemplo, en la lista

SUBROUTINE xy
SAVE A,B,/DATOS/
RETURN

```
SUBROUTINE Z
SAVE
```

la primera instrucción SAVE (SAVE A,B...) salva los valores de las variables A y B en el bloque de nombre DATOS, la segunda (SAVE) salva los valores de todas las variables utilizadas en la subrutina Z.

BLOCKDATA

En Fortran pueden introducirse subrutinas particulares que tienen la única función de definir e inicializar variables incluidas en un common con nombre (labelled common).
Además de las instrucciones de declaración, estas subrutinas pueden contener sólo instrucciones de inicialización. Por ejemplo en:

```
BLOCKDATA UNO
COMMON / B/V(5),C,X
DATA V (5*3.1)
END
```

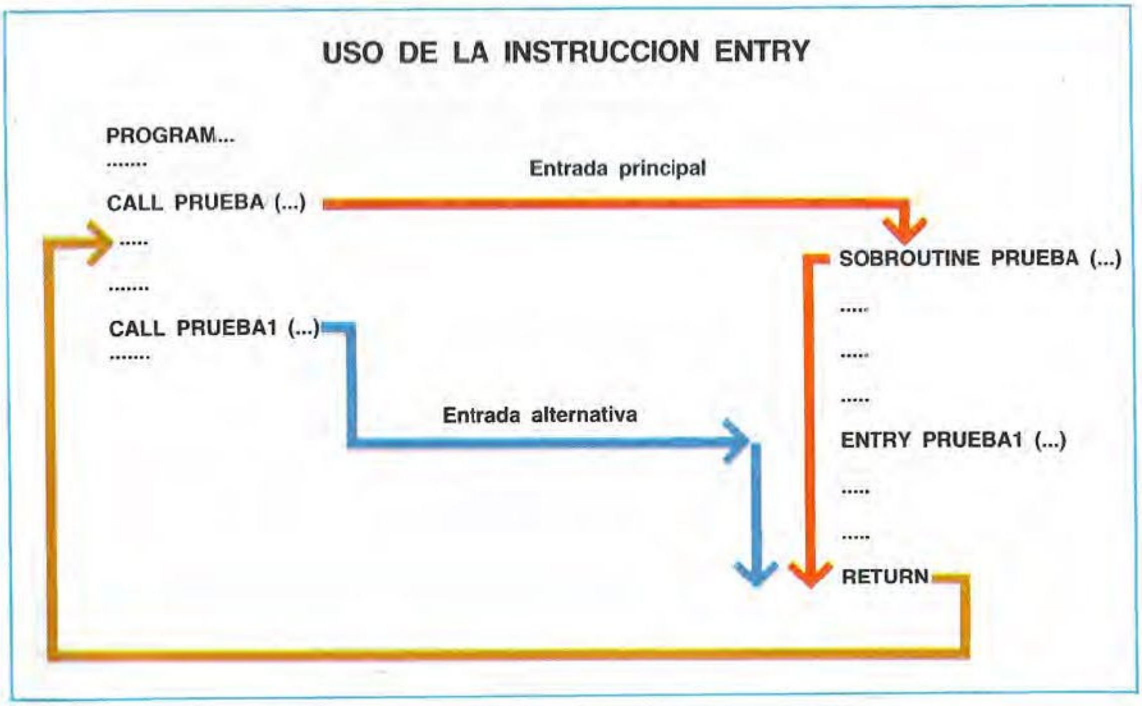
la subrutina de nombre UNO define un área de common de nombre B a la que pertenecen las variables V(5), C y X; además inicializa la matriz V(5) con valores 3.1 todos iguales.

Implantaciones particulares

Algunos Compiladores prevén notables implantaciones al Fortran estándar. A continuación se indican las principales (previstas en las normas MIL-STD-1753, Military Standard), que permiten acceder directamente a los bits de cada variable sencilla.

SHIFT	permite desplazar los bits un determinado número de posiciones
EXTRACCION	toma un determinado número de bits de una posición de memoria (o sea, permite la extracción parcial)
TEST	realiza un control sobre el estado (ON/OFF) de cada bit
SET	permite poner en ON un bit cualquiera
CLEAR	pone en OFF (cero) el bit especificado
MOVE	desplaza uno o más bits de una posición a otra
PRODUCTO LOGICO	función AND sobre los bits de una palabra
SUMA LOGICA	función OR
OR EXCLUSIVO	función XOR
COMPLEMENTO	función NOT

Estas funciones, propias de los Compiladores



más avanzados, son muy utilizadas en la gestión de los periféricos, en particular de los servomecanismos y de los dispositivos de adquisición de datos. Confieren al Fortran una elasticidad y variedad de empleo similares, si no superiores, a las del Basic; la única dificultad estriba en la necesidad de compilación, que hace poco ágil tanto el desarrollo como la comprobación del software.

Representación de las instrucciones en forma normal

La sintaxis de las instrucciones de cualquier lenguaje puede representarse en una forma gráfica que describe sus características.
Este tipo de representación llamada **Railroad Normal Form** (RNF), poco adecuada para la comprensión de un lenguaje por parte de un neófito, en cambio es muy útil para quien ya conoce por lo menos en líneas generales las instrucciones.
En principio, la representación de las instrucciones en forma normal puede utilizarse para cualquier lenguaje simbólico.

La representación consiste en ilustrar gráficamente las posibles alternativas para obtener un objetivo o para indicar una cierta función.
La única simbología particular está representada por un círculo con un número en su interior. Este símbolo significa que la correspondiente instrucción debe aparecer por lo menos un número de veces igual al indicado.
Por ejemplo, las instrucciones que pueden indicar un programa o una parte de él son

— Main (PROGRAM); debe aparecer por lo menos una vez, por lo que se indicará con el símbolo 1
— Function
— Subroutine
— Block data

Todas tienen significados análogos, y por tanto se representan de manera «paralela».
Un main puede contener instrucciones (program statement); análogamente, las funciones contienen function statement, etc.
Todas pueden tener labels (números de líneas) y el conjunto debe terminar con la palabra END.

Terminal remoto de Hewlett-Packard.



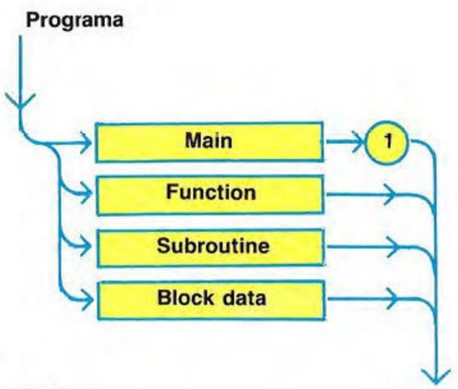
D. Bonecchi

Por tanto, para cada una de las voces anteriores se tienen nuevas posibilidades a un nivel de detalle mayor.
En el gráfico de esta página se han indicado muy sintéticamente los primeros dos niveles.

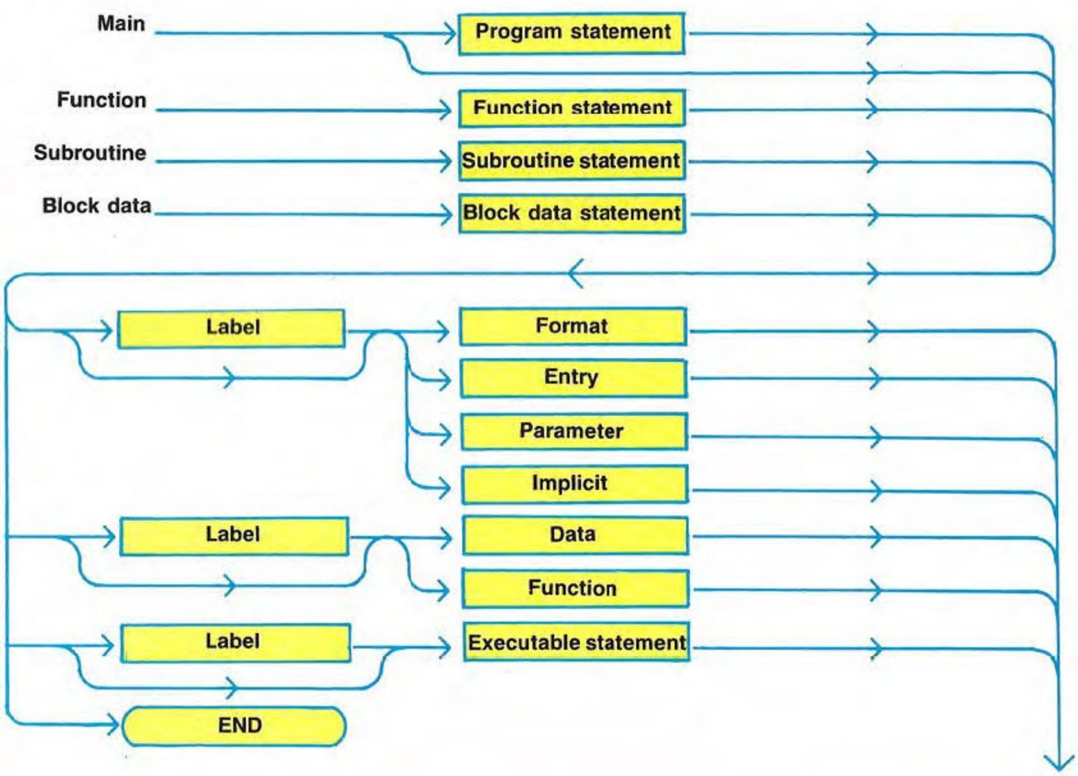
Procediendo hacia niveles de detalle mayores, se llega a la representación gráfica de cada instrucción sencilla.
En los gráficos de las páginas siguientes se han indicado algunos ejemplos.

REPRESENTACION RAILROAD NORMAL FORM

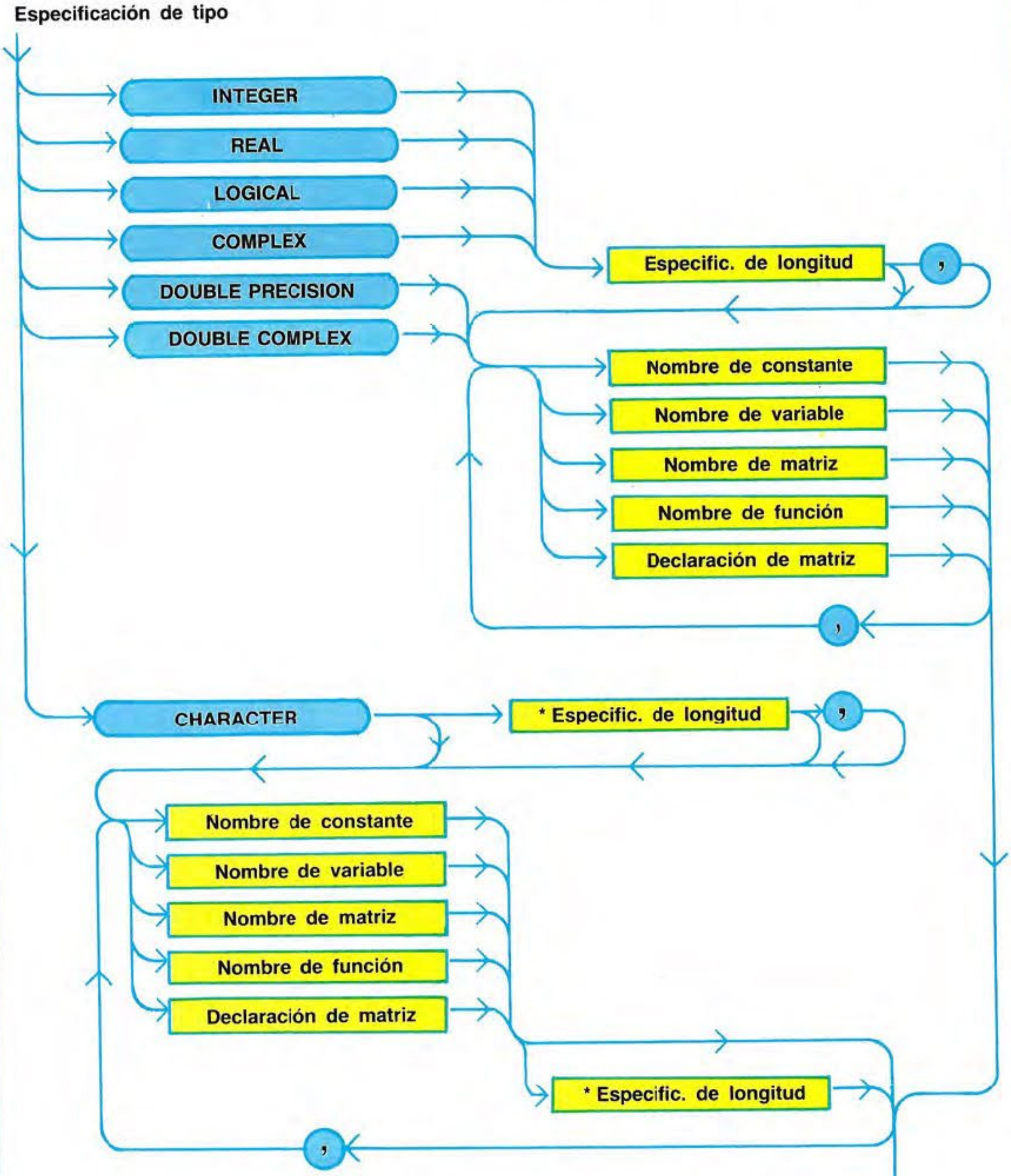
Primer nivel



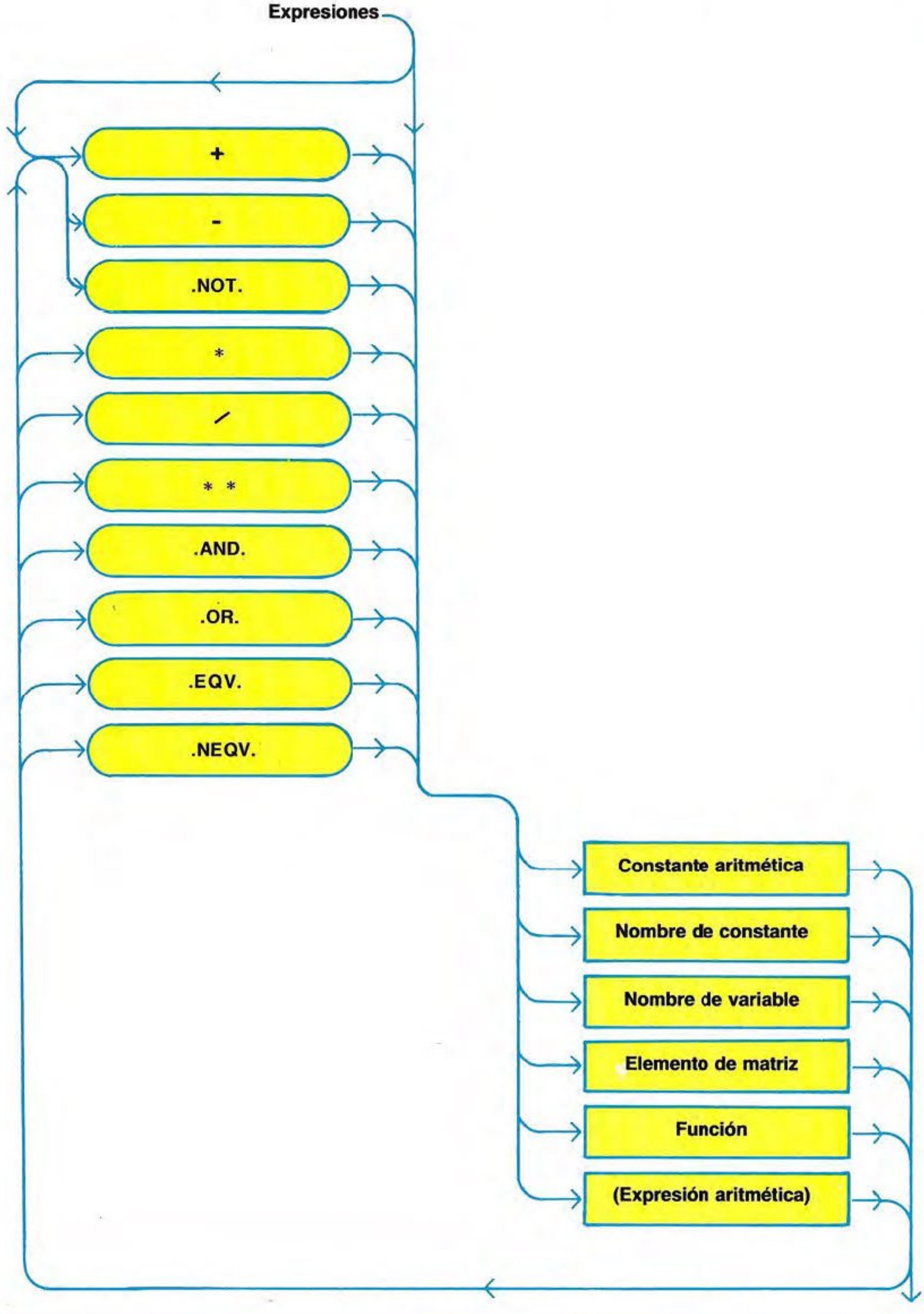
Segundo nivel



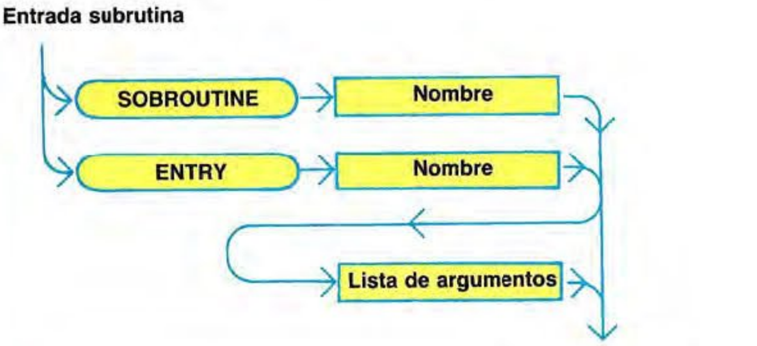
REPRESENTACION DE LA ESPECIFICACION DE TIPO



REPRESENTACION GRAFICA DE LAS EXPRESIONES

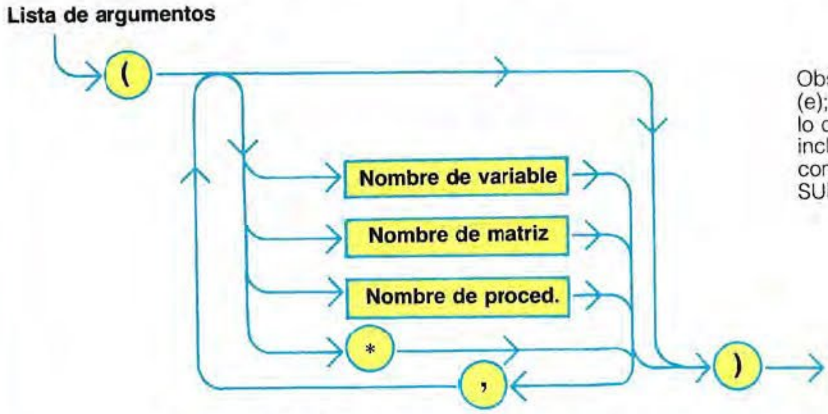


REPRESENTACION DE ENTRADA A SUBRUTINAS Y ESTRUCTURA DE PARAMETROS



Esta representación indica los dos modos de entrada en una subrutina. La lista de argumentos es facultativa y, por tanto, puede evitarse

Al nivel siguiente encontramos la sintaxis para la lista de los argumentos



Obsérvense los símbolos (e); indican que lo descrito debe incluirse entre paréntesis, como en el caso SUBROUTINE X (A,B,C)

Este símbolo, con flujo inverso, indica que los parámetros pueden ser múltiples (puede volverse directamente a las líneas de representación) y separados por una coma.
La línea continua que une los dos parámetros indica la posibilidad de utilizar subrutinas libres de parámetros, como en el caso SUBROUTINE xy().

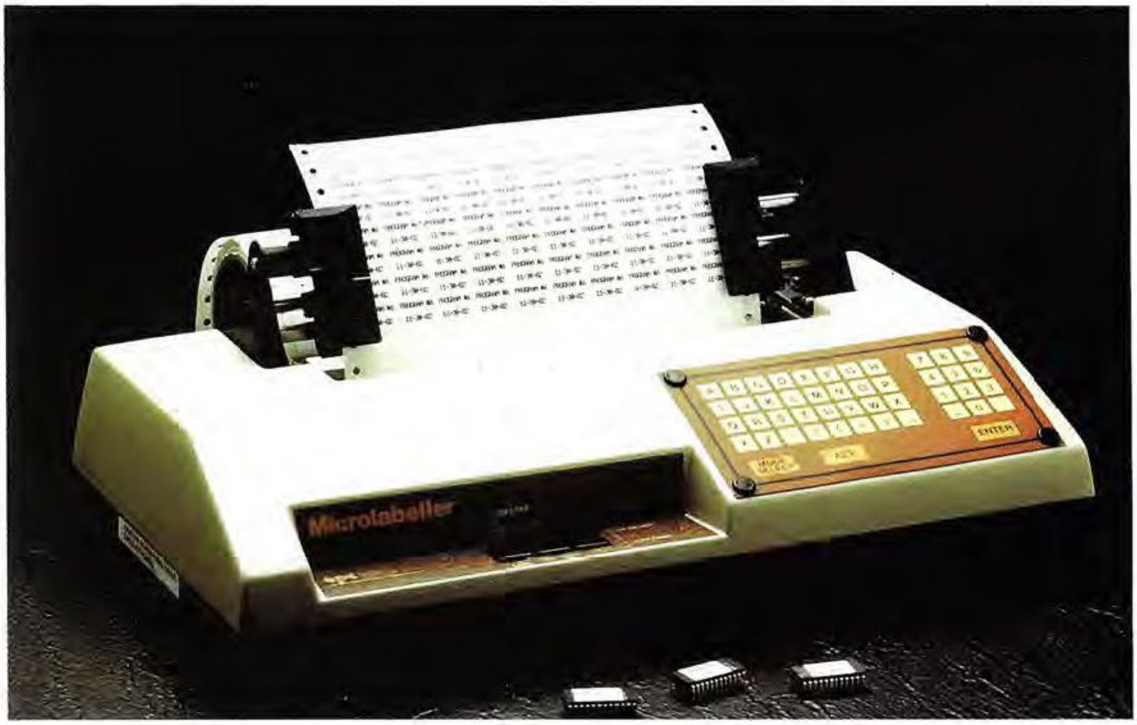
El lenguaje Pascal

A principios de los años 60, los calculadores eran máquinas de prestaciones limitadas, y para aprovechar de lleno las funcionalidades potenciales, el programador tenía que formular los algoritmos en lenguaje máquina.
Este pesado vínculo venía impuesto por las grandes limitaciones de la tecnología casi rudimentaria de entonces. Baste pensar que la capacidad máxima de la memoria no superaba los 16 a 32 kbytes. El programador estaba obligado a empujar la máquina al límite de sus prestaciones, buscando la optimización del rendimiento mediante el empleo de determinados «trucos» que terminaban por hacer prácticamente incomprensible cualquier programa. Con el progreso de la potencia de los procesadores, nacieron sucesivamente primero los lenguajes simbólicos ensambladores y después los de alto nivel, cada vez más orientados hacia el hombre y que comprenden una gran parte de las rutinas de empleo general, no previstas en las máquinas de las primeras generaciones.

Obviamente no fue fácil para los inventores del Fortran, introducido aproximadamente en 1957, forzar la resistencia y los prejuicios de los programadores, preocupados por la eficiencia de sus programas, en los cuales la «tecnología de los trucos» (trickology, en inglés) se había convertido en un arte. Después del Fortran, en 1960 siguió el Algol (Algorithmic language), que tendía a ser similar al Fortran, pero que introdujo también a nivel de instrucciones lo que con el Fortran sólo se tenía a nivel de expresiones aritméticas: la posibilidad de crear «estructuras» de sentencias, reagrupando las instrucciones de tal manera que formasen una operación completa. Así se podían componer las intrucciones en estructuras de control, de forma que las operaciones definidas por un programa podían ser vistas como una repetición de bloques, y no de sentencias sencillas. Desafortunadamente, el Algol, a pesar de su carga de innovaciones, nunca ha tenido un gran número de usuarios, y hoy sólo se emplea en ambientes de investigación y universitarios. El escaso uso de este lenguaje tanto en las aplicaciones técnicas como comerciales se deriva realmente de la incomprensión del concepto de estructura, que por otra parte hace difícil el uso de trucos y de saltos incondicionados dentro de las estructuras. La misma inercia que ha obstaculizado el paso del Assembler al Fortran hoy frena el empleo de los lenguajes que utilizan un estilo estructurado contra el estilo Fortran.

Actualmente, el problema ya no es hacer los ordenadores más veloces, sino poder organizar grandes y complejos programas asegurando los resultados deseados para todos los datos de entrada introducidos. Es decir, hoy es más importante hacer los programas fiables y fácilmente modificables en el tiempo. La filosofía de la programación estructurada es la de proporcionar al programador un método para afrontar los problemas de manera sistemática abandonando cualquier bagaje de trucos o empleo de secretos. En la tabla de la página siguiente se han relacionado algunos de los lenguajes más usados actualmente. Entre todos, sólo con el Pascal (y en estos últimos tiempos con el Ada) se ha vuelto a la estructuración propia del Algol. Sin embargo, la tendencia actual es la de dotar a los lenguajes existentes de estructuras de control manteniendo sus características originales. Por tanto, la estructuración ya no está vinculada al uso de un lenguaje particular en detrimento de los otros.

Impresora programable para la impresión de etiquetas.



B. Coleman/Marka

LENGUAJES DE PROGRAMACION

Año	Lenguaje	Descripción
1957	Fortran Formula translator	Emplea notaciones muy cercanas al álgebra, y la única estructura de datos utilizada es la matriz. Tiene la posibilidad de definir subrutinas como librerías. La elevada eficiencia de los primeros Compiladores hace aceptable el Fortran en todas las aplicaciones científicas, matemáticas y estadísticas. Es muy empleado.
1960	Algol Algorithmic language	Más conocido en Europa que en EE UU, introduce los conceptos bases de la programación estructurada. Piedra miliar del desarrollo del lenguaje de programación. Poco usado fuera de la investigación en software.
1960	Cobol Common business-oriented language	Es el lenguaje por definición en cualquier ambiente comercial. Utiliza una notación «english-like» (o sea muy cercana al lenguaje natural inglés). Es muy usado.
1961	Lisp List processing	Procesa datos no numéricos. La estructura de base es la «lista» cuyos elementos son datos elementales o listas. Es muy empleado en la investigación sobre la inteligencia artificial.
1962	Snobol	Lenguaje de programación para la manipulación de cadenas de caracteres. Incluye potentes instrucciones para la búsqueda de conjuntos de caracteres o cadenas. Utilizado, por ejemplo, en la gestión de bibliotecas.
1965	Basic Beginner's all-purpose symbolic instruction code	Originalmente concebido como simple lenguaje de fácil aprendizaje, hoy en día es el más utilizado en los ordenadores personales y microordenadores. Sin embargo, no existe un estándar, sino innumerables dialectos.
1965	PL/1 Programming language	Desarrollado por IBM debía ser el lenguaje que incluyera al mismo tiempo las características del Fortran, del Cobol y del Algol. Ha resultado un lenguaje extremadamente complejo y excesivamente extenso. Actualmente sólo se encuentra en máquinas IBM y es poco utilizado.
1967	Apl A programming language	Tiene un gran grupo de nuevas operaciones matemáticas, y además el usuario puede definir los propios operadores. Los programas Apl son muy concisos pero impenetrables.
1971	Pascal	Estudiado como instrumento para la enseñanza de la programación como disciplina metódica, incorpora las estructuras de control de la programación estructurada, muy eficientes en la redacción de grandes programas.
1980	Ada	Lenguaje en tiempo real, que puede emplearse en el control de procesos. Utiliza nuevas ideas sobre las estructuras modulares y las compilaciones separadas para el desarrollo de grandes proyectos.

La programación estructurada

La programación estructurada se basa en un concepto estructural: cualquier tarea que deba traducirse en un programa puede enfocarse con un trabajo de abstracción. Utilizando el lenguaje natural siempre es posible describir un problema en un conjunto de unidades fundamentales. El siguiente paso a realizar es subdividir cada unidad en otra subunidad y así hasta llegar al nivel de abstracción más bajo. El final de este trabajo de refinamiento conduce a un conjunto de instrucciones que pueden ser interpretadas por un calculador tanto si pertenece a un lenguaje de alto nivel, como el Fortran, como si se trata de instrucciones en lenguaje máquina. Por ejemplo, si hay que proyectar un procedimiento para preparar un banquete, pueden identificarse algunas operaciones fundamentales: adquirir los ingredientes necesarios, preparar los platos y servirlos. A su vez, cada uno de estos bloques puede subdividirse en otras unidades; así, la preparación de los platos podrá especificarse ulteriormente como trabajo de preparación de un conjunto de servicios (entremeses, primer plato, segundo plato, guarnición, postre, fruta). Obviamente, cada bloque puede detallarse en mayor grado hasta llegar a las operaciones elementales a realizar, como poner una olla de agua al fuego y esperar la ebullición. Es importante que a cada nivel de abstracción se empleen bloques conectados de manera sencilla, de acuerdo con el esquema general. Cada bloque ha de tener un solo punto de partida (inicio) y un solo punto final (salida). De este modo se gana ciertamente en claridad sobre las interconexiones entre los diversos bloques. La programación estructurada puede entenderse como un conjunto de técnicas para el proyecto y la codificación que guían al programador en la preparación de programas organizados, autodocumentados y fiables.

La implantación de los programas

La técnica de partir del problema general y subdivirlo en subproblemas más pequeños, a través de pasos sucesivos, se define como **top-down** (de arriba a abajo). Una segunda técnica utilizada es el **bottom-up** (del fondo hacia arriba) que prevé un enfoque opuesto al anterior: iniciar desde el nivel de detalle más bajo, formando pequeñas unidades que, a su vez, originan bloques más complejos.

En el caso de la preparación del banquete, la resolución del problema con la técnica bottom-up consistiría en preparar un determinado plato buscando los ingredientes que son necesarios, y en componer los diversos platos hasta llegar al banquete completo.

Utilizando la técnica estructurada, el desarrollo de programas puede dividirse en dos fases: **proyecto** e **implantación**; durante la fase de proyecto, el método de desarrollo más usado es el top-down, mientras que en la fase de implantación también puede ser útil el método bottom-up. En definitiva, para el análisis del problema, el sistema top-down es sin ninguna duda insustituible, mientras que en el caso de la síntesis (necesaria para transferir el programa a una particular máquina o lenguaje) el enfoque mejor es el bottom-up. En general, para desarrollar un programa de manera ordenada, son necesarios por lo menos cuatro niveles de abstracción antes de llegar al detalle en el lenguaje de programación elegido. Estos niveles son

1 / fin a alcanzar
2 / especificaciones
3 / seudocódigo
4 / código

Al escribir un programa existe un sencillo test para verificar la corrección de la implantación: describir de palabra el resultado que se quiere alcanzar. Si este paso resulta demasiado difícil, probablemente todavía no está bastante claro cuál es el objetivo final del programa, y por tanto es necesario volver a controlar las especificaciones. A continuación conviene examinar directamente si todo lo descrito es realizable con los recursos disponibles, tanto de hardware como de software, por ejemplo verificando que el calculador disponible sea adecuado en velocidad y capacidad de memoria. Definido de manera satisfactoria el objetivo a conseguir, deben puntualizarse las especificaciones, que pueden dividirse por lo menos en tres partes:

1 / datos de entrada
2 / procesos a realizar
3 / datos de salida

La completa descripción de los datos de entrada y de salida facilita la fase intermedia, o sea la descripción de las acciones a realizar en el interior del programa. Esta descripción (proceso a realizar) hecha en lenguaje natural evidencia

eventuales lagunas. Al redactar el seudocódigo, los bloques de entrada, cálculo y salida definidos anteriormente pueden dividirse en otros bloques en los que se usan estructuras que identifican una acción precisa. El seudocódigo no define cuál será el lenguaje final con que se escribirá el programa, pero el uso de estructuras de tipo Algol o Pascal ayuda a aclarar y a hacer legible todo lo que se quiere realizar. Un enfoque de este tipo proporciona asimismo la representación gráfica del programa que también puede sustituir los diagramas de flujo.

Al final de la preparación del seudocódigo puede realizarse un test importante, que es hacer leer el programa a una persona que esté fuera del proyecto. Si ésta puede interpretar correctamente su objeto, puede decirse se está en el buen camino para obtener un programa claro y de fácil manipulación.

El último paso que queda es traducir el seudocódigo, utilizando el lenguaje más adecuado para la aplicación particular. Algunos lenguajes, como el Pascal, hacen este objetivo muy fácil, dado que permiten definir procedimientos, funciones y variables booleanas, y permiten además añadir los comentarios para hacer el código similar al seudocódigo descriptivo. En esta fase es útil crear librerías propiamente dichas de procedimientos o funciones que puedan ser reclamados oportunamente en diferentes puntos del programa. Si el procedimiento se realiza correctamente, el programa principal (main) estará formado exclusivamente por pocas llamadas a procedimientos, y así sucesivamente, hasta llegar a los bloques finales que sólo contendrán unas pocas líneas de código. El programa así obtenido no tendrá necesidad de ulterior documentación. Efectivamente, la descripción del objetivo del programa (paso 1) es un óptimo resumen de sus características, la definición de las especificaciones da una buena descripción técnica y un seudocódigo bien presentado sustituye el diagrama de flujo.

El seudocódigo

Para ilustrar el método de obtención del seudocódigo y la forma de este último, supongamos que hay que resolver el siguiente problema: leer un número entero no negativo; si el número es igual a cero, el programa termina; de otro modo debe calcular e imprimir por separado las sumas de todos los números pares e impares leídos. El primer paso a realizar es definir de modo claro el objetivo del programa, que puede resumirse así: dado un valor entero positivo se analiza si este número vale cero, si es par o si es impar. Si el número leído es cero, el programa termina con la impresion

— del número de números pares entrados
— de su suma
— del número de números impares entrados
— de su suma

Si el número no es cero se actualizan el contador y la suma del grupo correspondiente, y después se pide un nuevo dato.

De este modo hemos descrito de manera exhaustiva cuál es el objetivo; a continuación podrán describirse las especificaciones del programa, examinando cuáles deben ser los datos de entrada, los procesos requeridos y los datos de salida, por ejemplo de la siguiente manera:

Datos de entrada	un número entero de valores comprendidos entre cero y el valor máximo (entero) representable en el procesador (32767 para una máquina de 16 bits)
Procesos	si el número vale cero, imprimir los resultados y terminar el proceso; de otro modo, identificar si el número es par o impar (diviéndolo por dos y controlando si el resto es cero). En base al resultado del test, se incrementa el contador de los números y se realiza la suma del grupo correspondiente. El proceso continúa pidiendo un nuevo dato
Datos de salida	impresión del número de los números pares introducidos y de su suma. Impresión del número de los números impares introducidos y de su suma

En esta fase hemos definido cuáles son los datos de entrada, los procesos necesarios (en forma bastante extensa) y los datos de salida.

La escritura del seudocódigo (nivel 3) entra en el detalle del programa utilizando el lenguaje na-

tural. Al lado se indica una forma de seudocódigo. Obsérvese el modo no usual de escribir las variables, cuyo nombre está constituido por dos partes unidas por un trazo (NUM-IMP).
Esta simbología permite utilizar una primera parte común y una segunda de especificación. Por ejemplo, las dos variables NUM-IMP y NUM-PAR se refieren a valores numéricos (NUM), pero una a los impares (IMP) y la otra a los pares (PAR).
Por tanto, sólo se trata de un método de identificación y no de una regla.

Los diagramas de flujo estructurados

El objeto del seudocódigo es análogo al del diagrama de flujo (figura de la pág. siguiente) y utilizando la indentación de las leyendas* es fácil seguir el programa. Haciendo uso de las palabras claves, como «inicio... fin» «hasta que... haz», «si... entonces... de otro modo», se identifican mejor los bloques del programa.
El pseudocódigo puede afinarse posteriormente, por ejemplo ampliando la parte anterior a la determinación si el número es par o impar. Normalmente esto no es necesario, puesto que puede entrarse en los detalles de este tipo directamente en la fase de detallado. Esta solución también permite optimizar el programa sobre el tipo de lenguaje y de máquina utilizados. En las figuras de las págs. 1209, 1210 y 1211 puede verse el programa codificado en Fortran 77. El código presentado no resuelve completamente el problema; por ejemplo, podría introducirse un tratamiento de los errores debidos a la introducción de datos de entrada errónea. Sin embargo, es claro que las eventuales modificaciones son fáciles de realizar, dados los criterios de modularidad con que se ha planteado el problema. Una ulterior y útil representación gráfica de los programas es la propuesta por Nassi y Schneiderman. Lo que se obtiene de esta representación es el llamado **diagrama N-S.**
En la figura de la pág. 1212 se ha representado el diagrama N-S del programa empleado como ejemplo. Este tipo de descripción evidencia el uso de las estructuras, que en un diagrama de flujo normal son menos evidentes. Por este motivo, los diagramas N-S también se definen como **diagramas de flujo estructurados**.

* Con el término indentación se entiende la diferente marginación de las diversas frases o palabras de descripción en el seudocódigo.

Ejemplo de seudocódigo

Definición constantes

MAXINT	iguales al máximo número entero aceptado por la máquina
ZERO	iguales al valor cero

Definición variables y su tipo

NUMERO	variables con el número leído; de tipo entero
NUM-PAR	número de valores pares leídos; de tipo entero
NUM-IMP	número de valores impares leídos; de tipo entero
SUMA-PAR	suma de los valores pares leídos; de tipo entero
SUMA-IMP	suma de los valores impares leídos; de tipo entero

```
Inicio
  Inicializa a cero las variables NUM-PAR,
  NUM-IMP, SUMA-PAR, SUMA-IMP
  Lee primero el valor de entrada → NU-
  MERO
  Hasta que NUMERO es diferente de
  ZERO haz
    Inicio
      Si NUMERO es par
      entonces
        Inicio
          Suma 1 a NUM-PAR
          Suma NUMERO a SUMA-PAR
        Fin
      de otra manera
        Inicio
          Suma 1 a NUM-IMP
          Suma NUMERO a SUMA-IMP
        Fin
      Lee nuevo valor de entrada →
      NUMERO
    Fin
  Leído un cero, imprime NUM-PAR,
  SUMA-PAR, NUM-IMP, SUMA-IMP
Fin
```


DIAGRAMA DE FLUJO
START
Definiciones e inicialización
Funciones I/O
Decisiones
Procesos
Definición constantes
Definición variables y su tipo
Inicialización variables
Lee el primer valor
¿Vale cero?
SI
NO
¿Es par?
NO
SI
Incrementa total números impares Suma el número a sus análogos
Incrementa total números pares Suma el número a los otros
Lee un número
Imprime los resultados
END

EJEMPLO DE PROGRAMACION ESTRUCTURADA EN FORTRAN

N. B. La codificación en FORTRAN ANSI 77 no permite utilizar en el interior de los nombres de variable el guión; por esto, NUM-IMP se codifica como NUMIMP

FORTRAN

PROGRAMADOR: CARLOS APARICIO — FECHA: MAYO 31, 84

PROGRAMA: ¿PARES O IMPARES? — Pagina 1 de 3

```
      PROGRAMA PARES IMPARES
      IMPLICIT NOMBRE
C     DEFINICION VARIABLES Y DE TIPO
C     NUMERO  --> VALOR LEIDO                           ENTERO
C     NUMPAR  --> NUMERO DE LOS VALORES PARES           ENTERO
C     NUMIMP  --> NUMERO DE LOS VALORES IMPARES         ENTERO
C     SUMAPAR --> SUMA DE LOS VALORES PARES             ENTERO
C     SUMAIMP --> SUMA DE LOS VALORES IMPARES           ENTERO
C
      INTEGER*2 NUMERO, NUMPAR, NUMIMP, SUMAPAR, SUMAIMP
C
C           INICIALIZACION A CERO DE LOS VARIABLES
            NUMPAR = Ø
            NUMIMP = Ø
            SUMAPAR = Ø
            SUMAIMP = Ø
C
C           LEE UN NUMERO
            PRINT *,' TECLEA UN NUMERO POSITIVO O NULO'
            READ *, NUMERO
```

FORTRAN

PROGRAMADOR CARLOS APARICIO FECHA MAYO 31,84

PROGRAMA ¿PARES O IMPARES Pagina 2 de 3

```
C     HASTA  QUE EL NUMERO NO ES IGUAL A CERO HAZ
C
      DO WHILE (NUMERO.NE.Ø)
         PRINT *,'EL NUMERO VALE',NUMERO
C
C        SI EL VALOR ES PAR: ANALIZADO POR LA FUNCION MOD QUE
C                            DA EL RESTO DE UNA DIVISION ENTRE ENTEROS
         IF (MOD(NUMERO,2).EQ.Ø) THEN
C
C        SI EL RESTO DE LA DIVISION ENTRE EL VALOR Y 2 VALE CERO
C                            ENTONCES EL NUMERO ES PAR
C
            NUMPAR = NUMPAR + 1
            SUMAPAR = SUMAPAR + NUMERO
C
C                                  DE OTRO MODO, EL NUMERO ES IMPAR
C
         ELSE
            NUMIMP=NUMINMP + 1
            SUMAIMP=SUMAIMP + NUMERO
```

FORTRAN

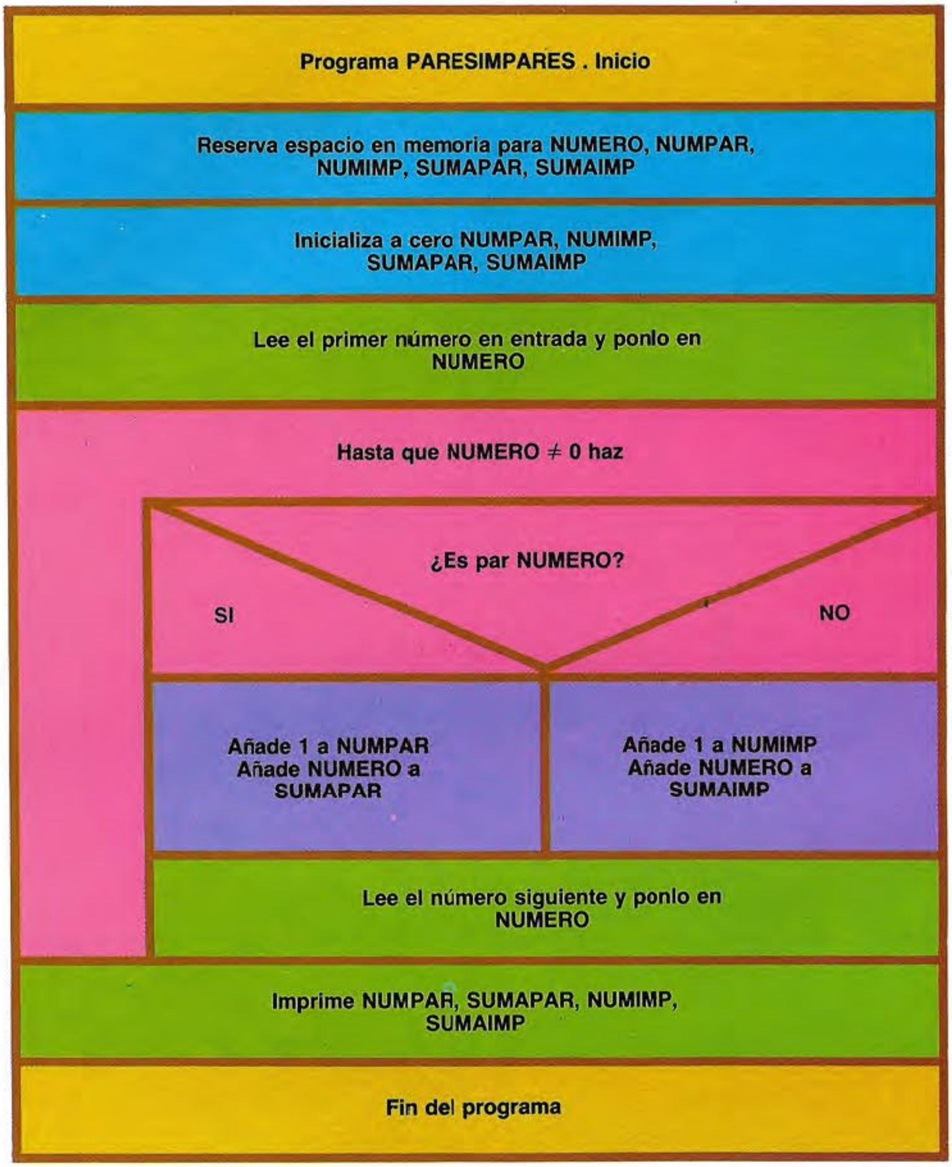
PROGRAMADOR CARLOS APARICIO FECHA MAYO 31,84

PROGRAMA ¿PARES O IMPARES

Pagina 3 de 3

```
            ENDIF
C
            PRINT *,'PULSA OTRO NUMERO O CERO(Ø) PARA SALIR'
            READ (*,END=99) NUMERO
C
         END DO
C
C        LEIDO UN CERO, IMPRIME LOS RESULTADOS
C
   99    PRINT *, 'NUMERO DE LOS VALORES PARES =',NUMPAR
         PRINT *, 'SUMA DE LOS VALORES PARES =',SUMAPAR
         PRINT *, 'NUMERO DE LOS VALORES IMPARES =',NUMIMP
         PRINT *, 'SUMA DE LOS VALORES IMPARES =',SUMAIMP
C
         STOP
C
      END
C
C     FIN DEL PROGRAMA PARES O IMPARES
```

DIAGRAMA DE FLUJO ESTRUCTURADO (DIAGRAMA N-S)



Cada bloque está representado por un símbolo particular.
En la figura de arriba, por ejemplo, pueden observarse las estructuras «hasta que... haz» y «si... entonces... de otra forma».
A continuación analizaremos, para los dos tipos de representación, los diagramas representativos de las operaciones fundamentales que componen un programa.

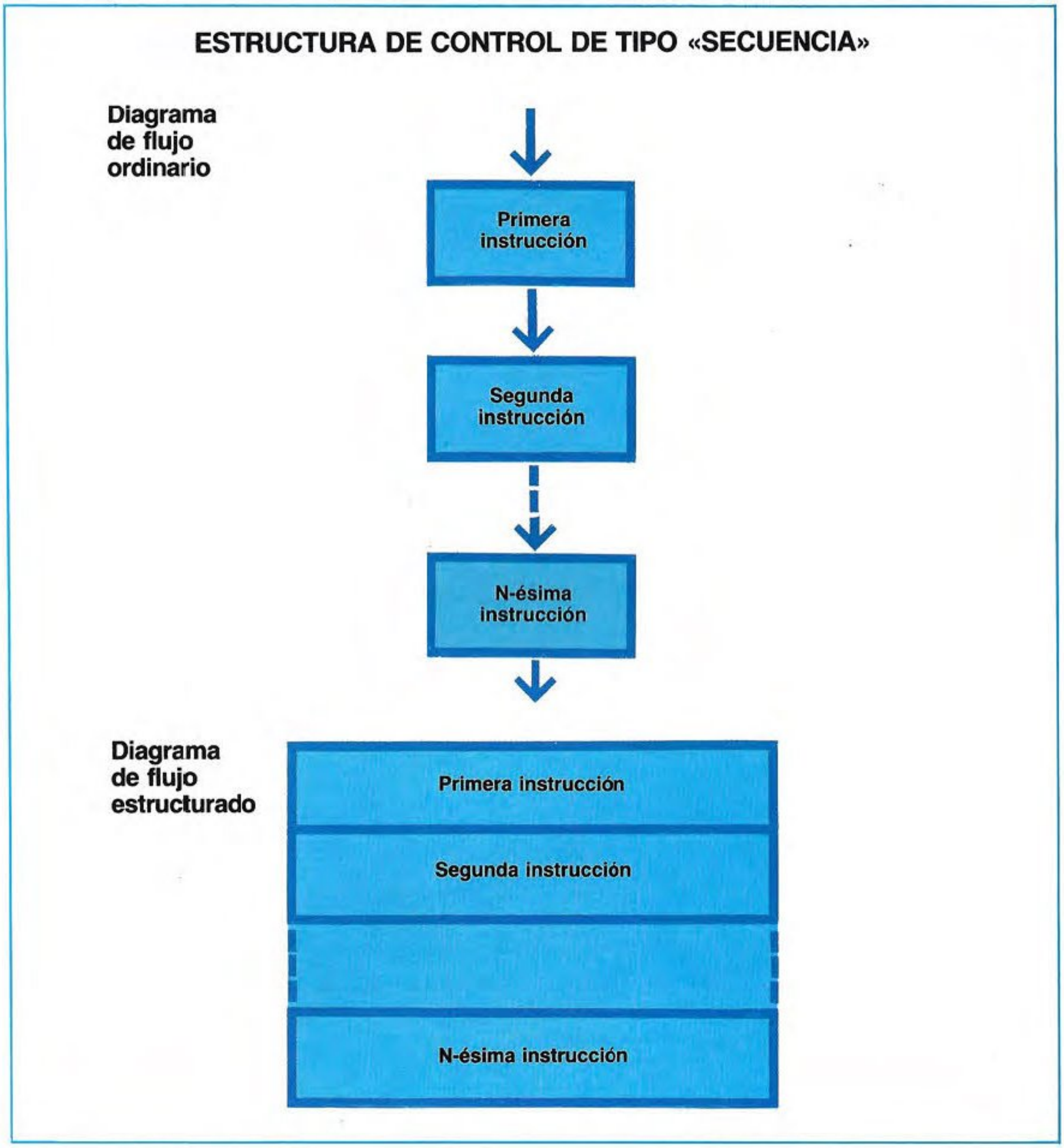
Las escrituras de control

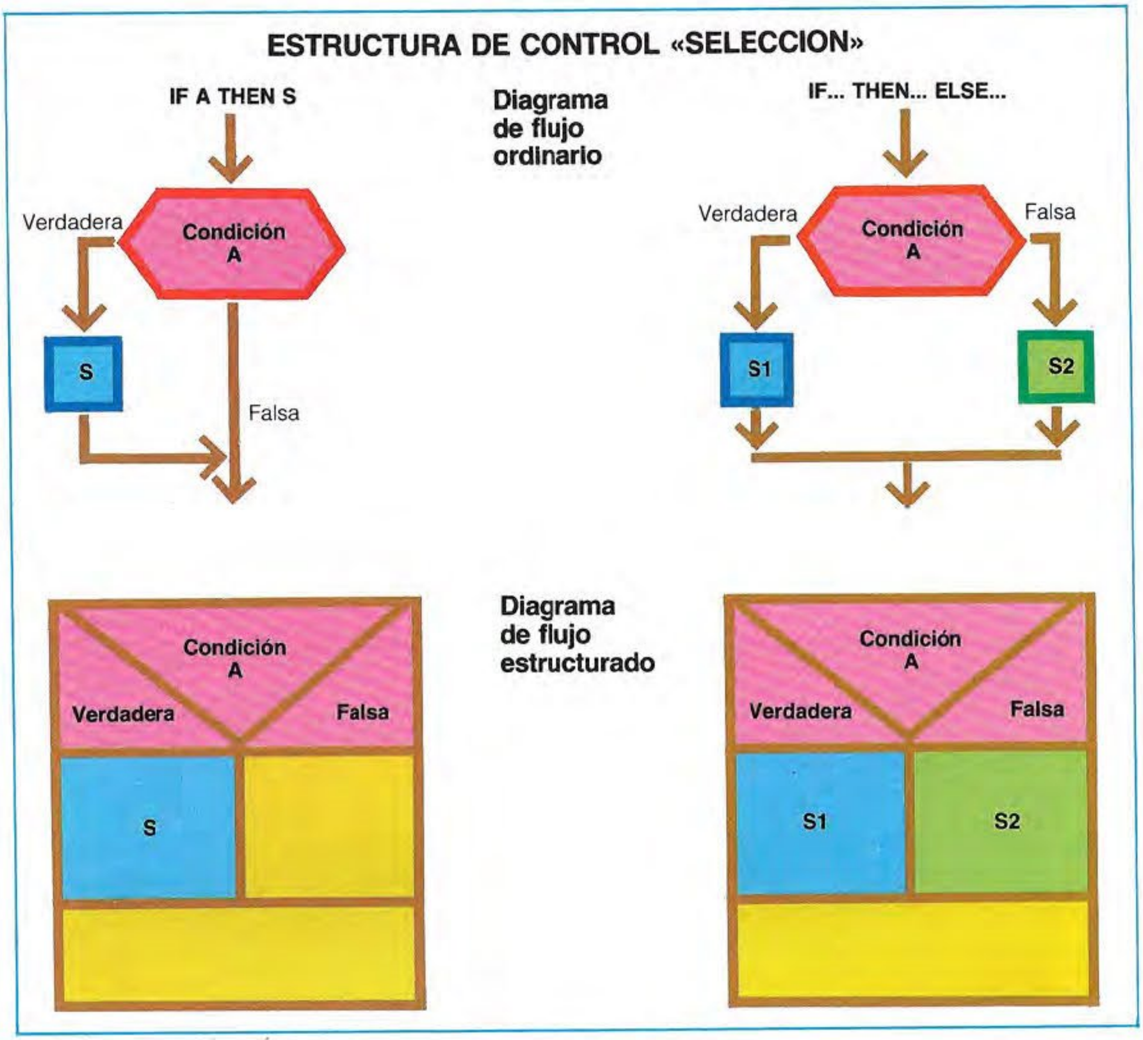
Las operaciones que se realizan en el interior de un programa pueden combinarse entre sí tantas veces como sea necesario utilizando tres estructuras fundamentales:

1 / secuencia
2 / selección
3 / repetición

Secuencia. La secuencia es la escritura de base más recurrente en un programa, y consiste en una sucesión de operaciones a realizar una después de otra. En la figura de abajo se han ilustrado los dos tipos de representación. El nivel de detalle con que se definen los contenidos de los bloques depende del nivel de afinación que se quiere obtener (al límite, todo un programa puede representarse por un solo bloque).
En algunos lenguajes, el principio y el fin de una secuencia se evidencian con las dos palabras clave BEGIN y END (es el caso del Algol y del Pascal). En otros, el principio y el fin de una secuencia se identifican implícitamente, ya que ésta se cierra entre dos palabras clave que pertenecen a una estructura diferente.

Selección. Dos secuencias se dicen **combinadas en selección** cuando con el cumplimiento o no de una condición se ejecuta una u otra. Por ejemplo, en los diagramas de la derecha de la página siguiente, según el resultado se realiza la secuencia S1 o la secuencia S2.
En casi todos los lenguajes de programación existe la sentencia IF... THEN... ELSE, o sea SI... ENTONCES... DE OTRA FORMA. Éste es un





ejemplo de instrucción compuesta. En el Fortran 77, entre THEN y ELSE puede insertarse más de una instrucción (como en el programa anterior), por tanto puede haber una secuencia tanto después de la palabra clave THEN, ELSE como después de la ELSE y el final de la selección (palabra clave ENDIF). Una secuencia también puede no contener ninguna instrucción en la palabra clave ELSE (se dice que contiene la instrucción nula; ver arriba a la izquierda), con el significado: «si la condición no es verdadera prosigue directamente». En el Cobol se admiten sólo las secuencias formadas por una sola sentencia, mientras que en Pascal y en Algol la secuencia de más instrucciones siempre va precedida de la palabra clave BEGIN y cerrada con la palabra clave END; las dos palabras clave tienen un empleo análogo al de los paréntesis. Por ejemplo, las dos estructuras

```
IF condición THEN
   Instrucción 1
Instrucción 2

IF condición THEN
   BEGIN
           Instrucción 1
           Instrucción 2
   END
```

son completamente diferentes.

Si la condición es verdadera, en el primer caso se ejecuta sólo la primera sentencia, mientras que en el segundo se realiza todo lo encerrado entre los delimitadores BEGIN y END.

Repetición. Las estructuras de repetición son fundamentalmente cuando la operación global a realizar está constituida por una sucesión finita

pero no siempre conocida a priori de operaciones iguales. Con esta técnica es posible utilizar más veces una misma secuencia de instrucciones. Las estructuras de control de repetición fundamentales son dos: WHILE... DO..., o sea HASTA QUE... HAZ..., y REPEAT... UNTIL..., o sea REPITE... HASTA QUE... Las dos estructuras se ilustran en la figura de abajo y en la de la página siguiente. Si bien a primera vista pueden parecer iguales, conceptualmente son diferentes. En la WHILE... DO... se efectúa el test en la condición y la secuencia S se realiza sólo si la condición resulta verdadera. En la REPEAT... UNTIL... se realizan primero la secuencia S y después el test: si la condición resulta falsa, la secuencia se repite; de otro modo se sale de la estructura. En Pascal y en Algol, si la secuencia está compuesta por más de una instrucción, son necesarios el paréntesis BEGIN... END en la WHILE... DO... En cambio no son necesarios en la REPEAT... UNTIL..., puesto que el principio de la secuencia está definido por la palabra clave REPEAT, y el final por la palabra clave UNTIL. Por el contrario, en Fortran 77, la secuencia se cierra después de DO... WHILE... y ENDDO. En este lenguaje no existe una verdadera forma REPEAT... UNTIL..., que puede sustituirse por la instrucción

```
S1 Instrucción
    IF (.NOT. condición) GOTO S1
```

o en otros casos con una simple DO... ENDDO.

ESTRUCTURA DE CONTROL «REPETICION» (WHILE... DO...)

Diagrama de flujo ordinario

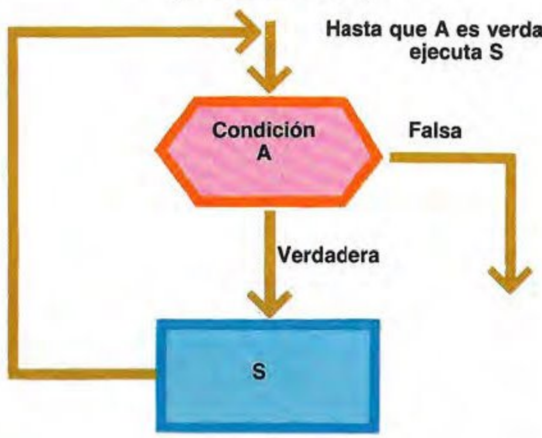
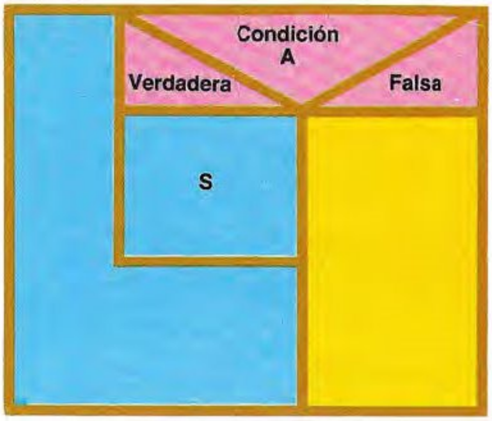
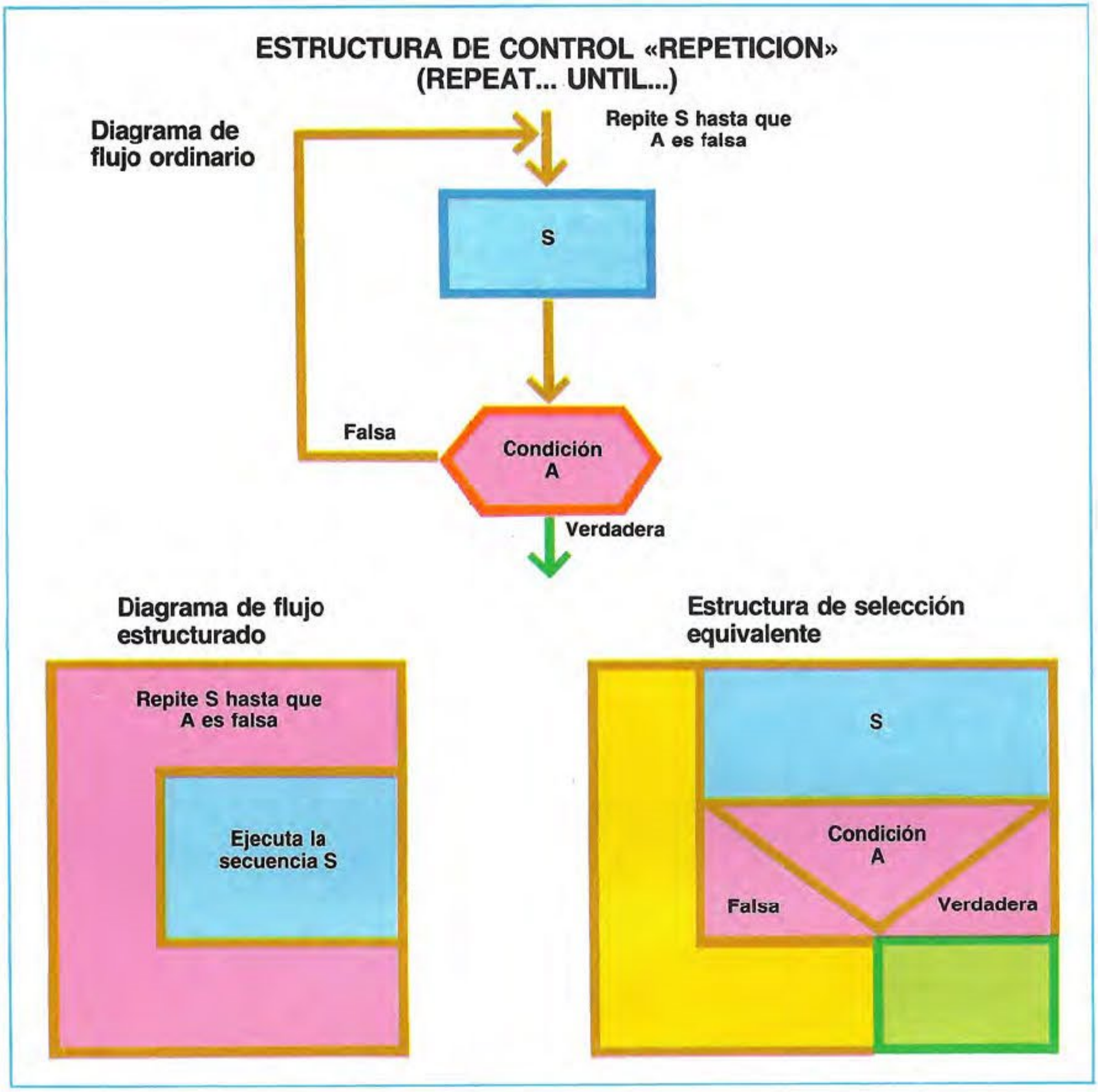


Diagrama de flujo estructurado



Estructura de selección equivalente





En Cobol es posible definir tanto la estructura WHILE... DO..., de la manera

PERFORM nombre-de-procedimiento
UNTIL expresión

como la estructura REPEAT... UNTIL..., de la manera

Inicialización de expresión con FALSE
PERFORM nombre-de-procedimiento
UNTIL expresión

Las estructuras vistas constituyen el núcleo principal de la programación estructurada. Existen otros, pero puede demostrarse que las descritas son suficientes para describir cualquier algoritmo, y esto es cierto incluso limitándose a las estructuras de secuencia IF... THEN... ELSE... y ELSE... DO..., puesto que las demás pueden obtenerse como composición de éstas.

Los datos en el Pascal

Un programa escrito en Pascal normalmente está dividido en dos partes principales: la declaración de los datos y el conjunto de las instrucciones. La primera parte define los tipos de datos que utilizará el Compilador, pero constituye también un utilísimo medio para mejorar la legibilidad del programa. La segunda contiene las instrucciones que definen el modo en que se procesarán los datos especificados anteriormente.

Tipos de datos

Una de las novedades más significativas introducidas por el Pascal es la formalización del concepto de **tipo de dato**. Se dice que determinadas constantes o variables pertenecen al mismo tipo si tienen características iguales entre sí y están sometidas a procesos del mismo modo. Por ejemplo, el conjunto de los números enteros, o sea, los números sin coma, constituye el tipo de dato entero.

En el Pascal, los tipos de datos elementales están definidos como **escalares**. Los tipos escalares no pueden descomponerse en tipos más sencillos; en cambio, pueden combinarse de forma diferente para formar los **tipos estructurados**. Por ejemplo, un vector de números enteros de diez elementos

IA(I) I = 1,..., 10

es un dato de tipo estructurado, puesto que está formado por diez datos escalares IA(I), IA(2),..., IA(10).

Los datos escalares se subdividen en **tipos estándar** y **tipos definidos por el usuario.**

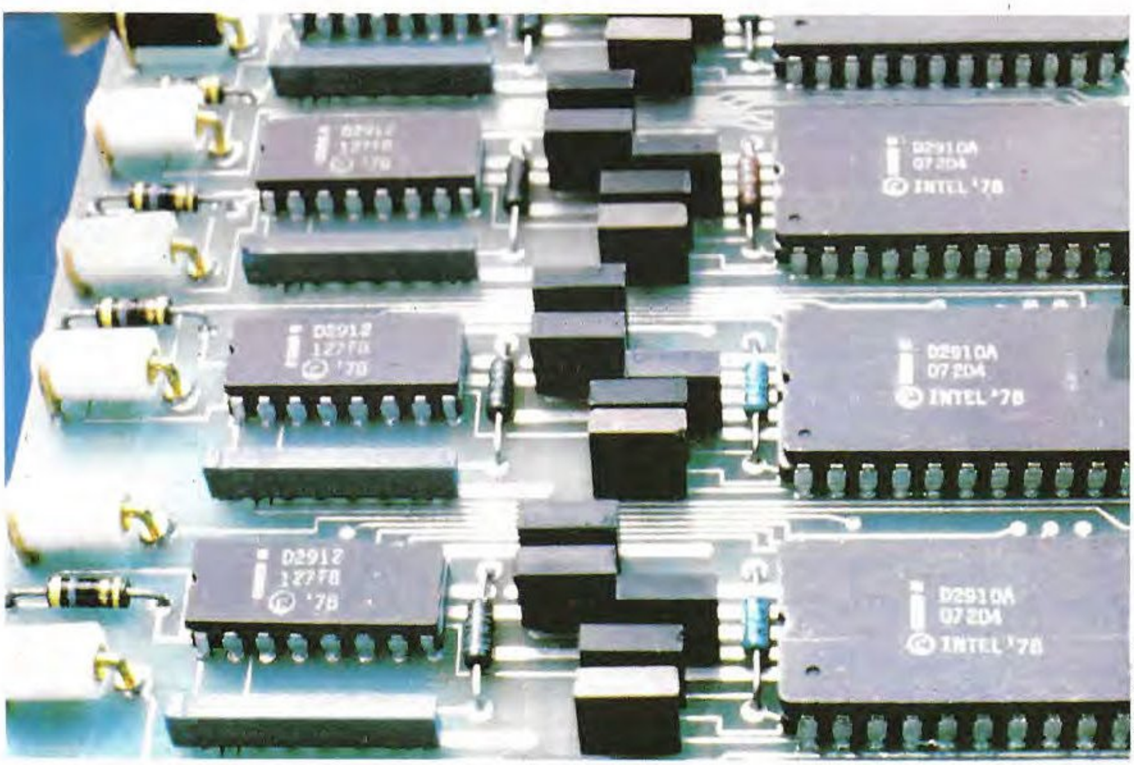
Los tipos estándar son tipos predefinidos por el Pascal. Son de tipo entero, real, carácter, booleano y pueden utilizarse sin especificar el campo de validez porque este campo lo establece automáticamente el Compilador Pascal.

Los tipos de datos definidos por el usuario representan una característica del Pascal, que permite al usuario crear ad hoc tipos particulares. El usuario establece el campo de validez y los elementos que le pertenecen para resolver determinados problemas.

En cambio, los tipos de datos estructurados son el tipo **matriz**, término genérico con que se indican los vectores y las matrices de dos o más dimensiones, el tipo **set**, o sea un conjunto de elementos del todo similar a la matriz pero que se procesa como un único elemento, y el tipo **file**, con el significado de conjunto de records. En la figura de la página siguiente se muestra una tabla sinóptica que representa todos los tipos de datos admitidos por el Pascal.

El tipo entero. Los datos de tipo entero son los que pertenecen al conjunto de los números naturales (sin coma) precedidos o no del signo positivo o negativo. El límite inferior y superior de este conjunto, o sea el número negativo más pe-

Conjunto de circuitos integrados montados en la tarjeta madre de un ordenador.



J. Pickerell/Marka

TIPOS DE DATOS ADMITIDOS POR EL PASCAL



queño y el positivo más grande, dependen evidentemente de la capacidad del calculador utilizado. Algunas grandes máquinas (palabra de 36 bits) admiten para los números enteros una extensión de -2^{35} a 2^{35} -1, o sea entre $-34\,359\,738\,368$ a $+34\,359\,738\,367$. Los ordenadores personales suelen utilizar 16 bits para la representación de los números enteros, por lo que se tiene una extensión de los valores de $-32\,768$ a $+32\,767$.

Para evitar los problemas de superación de la capacidad de calculador, especialmente cuando los programas se transportan de una máquina a otra, el Compilador Pascal predefine y utiliza una variable entera, llamada MAXINT, que contiene el número entero más grande utilizable en el calculador que se adopta. El conocimiento del valor de esta variable permite prescindir del conocimiento de la longitud de la palabra de máquina del calculador utilizado. En la primera figura de la página siguiente se muestra, según una notación particular llamada **diagrama sintáctico**, la sintaxis formal de la escritura de un dato de tipo entero. Esta notación, que de ahora en adelante se utilizará a menudo, sirve para expresar de forma compacta las reglas sintácticas del Pascal. Ésta indica, de manera totalmente análoga a la notación de los diagramas de flujo, cómo construir un tipo de dato o una instrucción combinando de manera oportuna, en los diversos campos, los caracteres alfanuméricos y los símbolos de separación introducidos.

Según este tipo de sintaxis, resultan válidas todas aquellas proposiciones (números e instruc-

ciones) que pueden generarse partiendo de la izquierda del diagrama, según las flechas y saliendo a la derecha. Por ejemplo, dado el diagrama sintáctico visible en la figura de más abajo de esta página, resultan sintácticamente válidos los números

1
1 2 3
1 2 1
1 2 1 2 3

mientras que no resulta correcto el número

1 3

puesto que en el diagrama no existe ningún camino que permita ir de izquierda a derecha pasando sólo por los números 1 y 3.
Análogamente, no resultan sintácticamente válidos los números

1 2
2 3
1 2 1 3
1 2 3 3

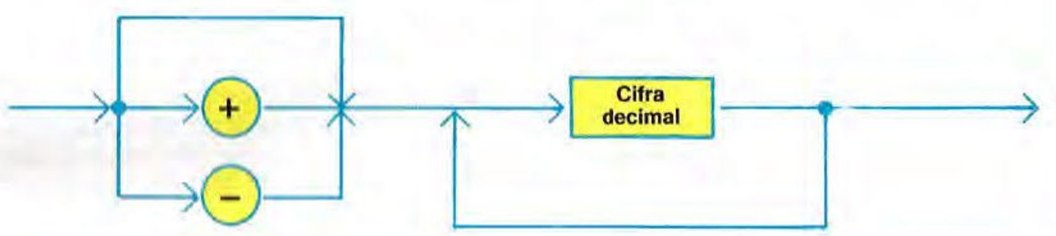
Volviendo al diagrama de la primera figura de abajo, se observa que el tipo de dato entero está constituido por un número de cifras decimales de la longitud que se desee (en los límites, evidentemente, de la capacidad del calculador). El conjunto de las cifras va precedido por un símbolo +, –, o ninguno, según el camino elegido en las tres ramas del diagrama.
Siguiendo este tipo de notación, algunos ejemplos válidos de enteros son los siguientes:

16384
–14
+250
0

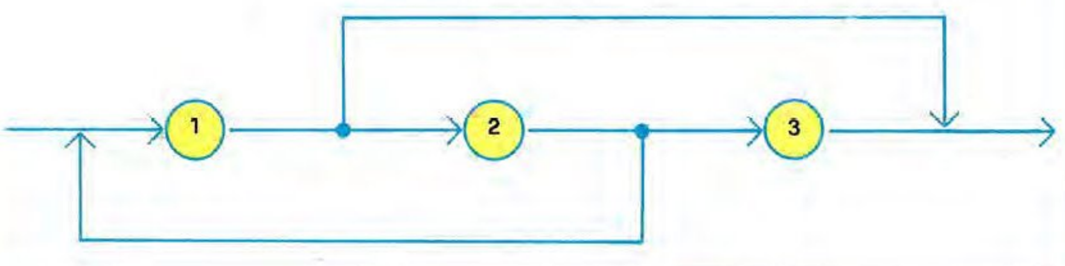
Las operaciones admitidas en el Pascal sobre números enteros son:

1 / Operaciones aritméticas (suma, resta, multiplicación)
2 / Operaciones de división DIV y MOD, con los significados que siguen
DIV división entre enteros, sin la parte fraccionaria; por ejemplo, 29 DIV 8 = 3
MOD resto de la división entre enteros; por ejemplo 29 MOD 8 = 5
3 / Operaciones relacionales
> mayor
= igual
< menor
> = mayor o igual
< = menor o igual
< > no igual (diferente)

SINTAXIS DEL TIPO ENTERO



EJEMPLO DE DIAGRAMA SINTACTICO



A diferencia de los dos primeros grupos de operaciones, el resultado de una operación relacional entre dos números enteros no es un número entero, sino una variable booleana, que puede asumir el valor «verdadero» o «falso».
Finalmente hay algunas funciones que actúan tanto sobre números enteros como reales y proporcionan el resultado entero. Éstas son:

ABS (I)	proporciona el valor absoluto del número entero I, o sea el número I sin signo
SQR (I)	elevación al cuadrado del número I
ROUND (R)	proporciona el número entero que más se aproxima al número real R; por ejemplo ROUND (4.87) = 5
TRUNC (R)	proporciona la parte entera de R; por ejemplo TRUNC (4.87) = 4

El tipo real. El tipo real se refiere al conjunto de los números que tienen una parte entera y una fraccionaria. Pueden representarse en la **notación decimal** corriente, con por lo menos una cifra antes y después del punto decimal, o bien en la **notación exponencial** o científica. Esta última está constituida por una mantisa compuesta por un número decimal con parte entera de por lo menos una cifra, seguida de la letra E y de la potencia entera de 10 que, multiplicada por la mantisa, proporciona el valor del número real (de forma del todo análoga al Fortran y al Basic). Por ejemplo, el número 1.237E03 equivale a 1.237×10^3 o sea a 1237.
La sintaxis completa de la representación de los datos reales se muestra en la figura de abajo. Siguiendo el diagrama sintáctico puede comprobarse la validez de los números

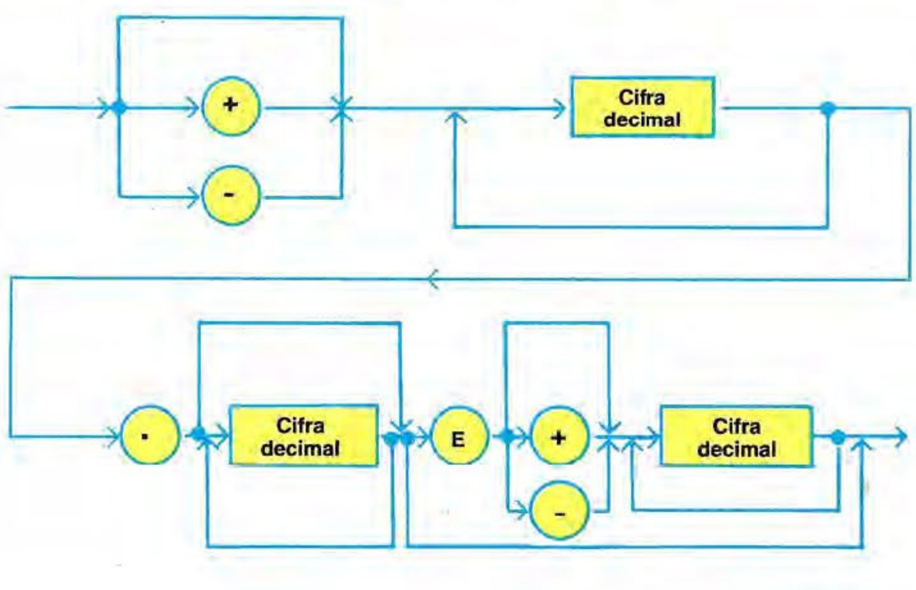
–32.1
+8.3E – 9
–42.0
–6.9E – 19

y la no validez de los números

14	(número entero)
027	(ninguna cifra antes del punto decimal)
8.57E + .2	(no se admite la potencia decimal)

Como para los primeros enteros, también para los reales existen limitaciones para el conjunto de los números representables. Estas limitacio-

SINTAXIS DE TIPO REAL



SINTAXIS DE TIPO CARACTER



nes se refieren tanto a la precisión, o sea al número de las cifras introducidas después del punto decimal, como a los valores enteros representables.

Las operaciones admitidas sobre números reales son

1 / las cuatro operaciones algebraicas + –, ∗, /
2 / las operaciones relacionales, de manera idéntica a lo que se ha visto para los enteros.

Además se admiten las siguientes funciones, cuyo resultado es un número real:

ABS(R)	valor absoluto del número R
SQR(R)	cuadrado de R
SQRT(R)	raíz cuadrada de R
LU(R)	logaritmo en base e de R
EXP(R)	e elevado a R
SIN(R)	seno de R
COS(R)	coseno de R
ARCTAN(R)	arcotangente de R

El tipo carácter. En Pascal, el tipo carácter puede incluir todos los símbolos alfanuméricos y ortográficos representables en el calculador utilizado. A causa de las diferencias existentes entre los símbolos disponibles en cada máquina, el conjunto de los datos de tipo carácter no siempre es el mismo (para número y para composición) al variar la máquina. Estas diferencias deben tenerse en cuenta cuando se transportan los programas de un calculador a otro. Sin embargo, puede contarse con un conjunto mínimo de caracteres disponibles con seguridad en cada calculador. Este conjunto mínimo contiene las 26 letras mayúsculas de la A a la Z (todo el alfabeto inglés), las cifras de 0 a 9 y los caracteres especiales que indican los signos ortográficos y los operadores, además del espacio. La sintaxis de escritura de los datos de tipo carácter se muestra en la figura de arriba. Como se ve, un dato de tipo carácter siempre está encerrado entre dos comillas, como en los ejemplos

'P'
'7'
' '
'
' '

El tipo carácter no debe confundirse con el tipo cadena que, al ser un conjunto de caracteres, es un dato de tipo estructurado, es decir, compuesto por un conjunto de datos de tipo escalar. Por ejemplo, la cadena

'PROGRAMA'

está compuesta por los caracteres sencillos 'P' 'R' 'O' 'G' 'R' 'A' 'M' 'A'.

El tipo estructurado cadena se describirá detalladamente más adelante. Las funciones que operan sobre datos de tipo carácter son:

CHR (I)	Restituye el carácter correspondiente al número I en la codificación interna de los caracteres en el calculador. I es un entero comprendido entre 0 y 128 (o 255).
ORD ('C')	Es la función inversa de la CHR (I). Restituye un número entero correspondiente a la representación interna del carácter C.
PRED ('C')	Función predecesora. Restituye el carácter inmediatamente precedente al carácter C en el juego de los caracteres válidos. Por ejemplo, PRED ('B') es igual a 'A'.
SUCC ('C')	Función sucesora. Restituye el carácter inmediatamente siguiente al carácter C; por ejemplo, SUCC ('B') es igual a 'C'.

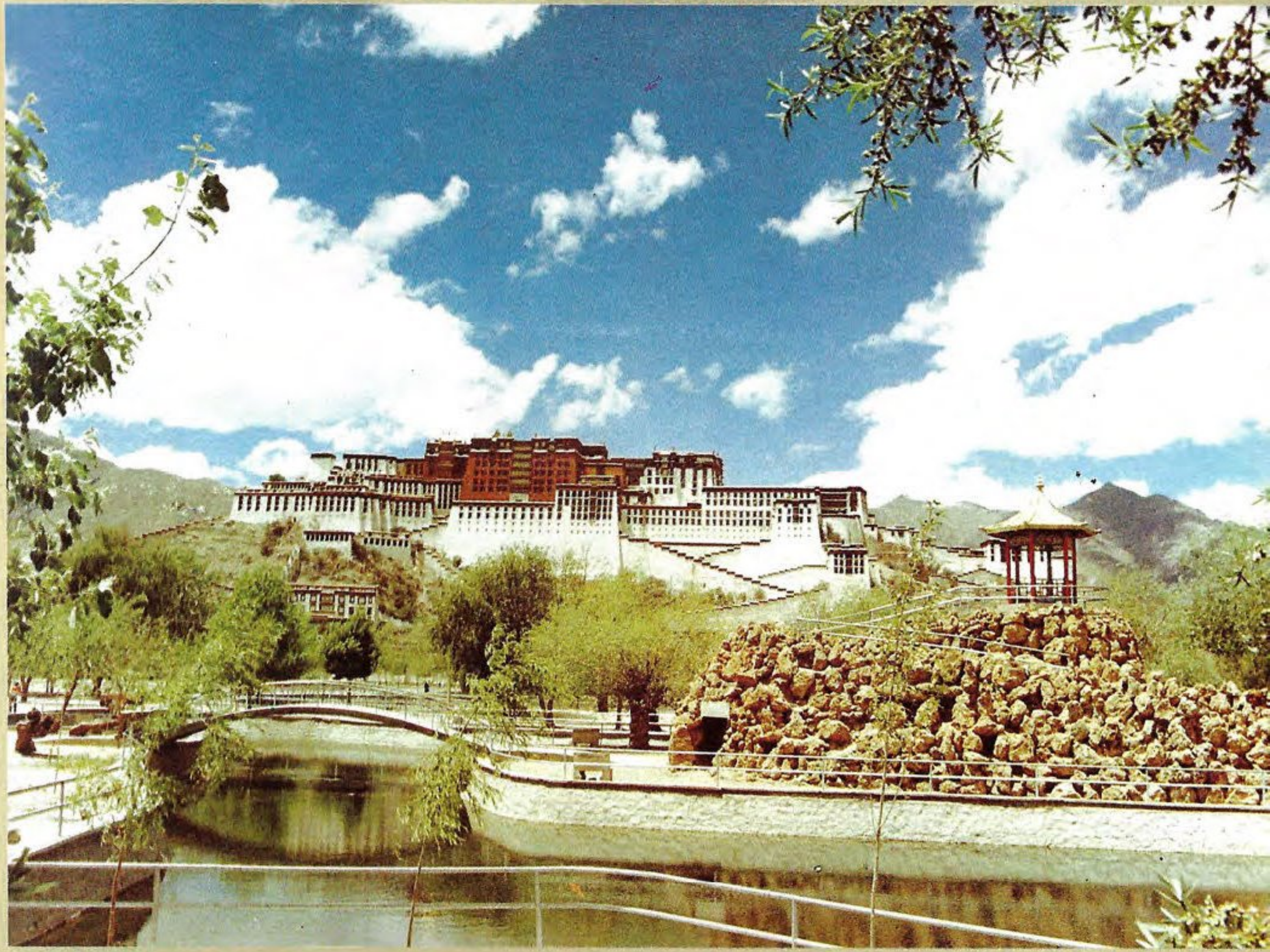
Es interesante observar que estas dos últimas funciones pueden aplicarse también a datos de tipo entero; por ejemplo,

PRED (5) = 4
SUCC (5) = 6

Cómo se proyecta un videojuego inteligente (1)

El éxito comercial de los videojuegos es indiscutible: si bien hoy se van redimensionando, o por mejor decirlo estabilizando a niveles fisiológicos que verosímilmente se mantendrán por largo tiempo, permanecen para siempre como un dato en nuestras costumbres sociales.
Este fenómeno habría podido ser efímero y de breve duración si el sistema industrial que está en sus orígenes no se hubiese puesto a producir una enorme cantidad de temas y vicisitudes que satisfacen y motivan continuamente la curiosidad y la sed de aventuras del jugador de vídeo. En realidad, la actitud de éste con respecto al contenido cultural y educativo de los productos comerciales que se le ofrecen no ha sido y todavía no es suficientemente crítico.
Antonio Berga, un experto agudo y sensible de la cultura informática del juego, sugiere en estas páginas un modo para acercar de manera correcta los contenidos culturales, además de los recreativos, de un juego milenario: el Go

Encontramos expresiones significativas de la cultura del juego en todas las partes de nuestro planeta: éste es un patrimonio común en el espíritu de los pueblos, una especie de lengua universal que a menudo facilita la comprensión de la filosofía de la vida y de las costumbres sociales de poblaciones lejanísimas entre sí, incluso culturalmente.
Es necesario acercarse a este intercambio de experiencias culturales de manera leal, sin trastornar los contenidos y poniendo además la máxima atención a la lectura de los significados, que siempre que sea posible no deben cifrarse con el propio filtro cultural.
Ciertamente, todo esto es difícil o casi imposible, pero en el esfuerzo de apertura que activa el intercambio está el valor formativo de la experiencia. Cuando nos proponemos jugar a través de un instrumento recreativo que proviene de otros confines de nuestra cultura es fundamental no banalizarlo, o sea no convertirlo en un mero ejercicio de virtuosismo con el joystick, a través del uso del procesador.
Una elección de campo que puede aplicarse también a través de una sabia selección del juego a que se refiere, es hoy, en la época del juego electrónico de masas, en el período en que la ilusión de poder hacerse rico vendiendo el «game» está muy difundida, los lectores podrían ser estimulados a leer con sagacidad dentro de sí mismos, a la búsqueda de los propios deseos más verdaderos y genuinos, intentando transferirlos después a la propia vida sensible, expresados en forma de juegos.
Sabemos que existe una rica y casi infinita variedad de juegos. También sabemos que éstos son a menudo los portaestandartes del espíritu de un pueblo y sabemos que este significado es tenaz. A veces, el juego se convierte en la expresión más evidente de la supervivencia en los acontecimientos sociales de las tradiciones, de los usos y de las costumbres de pequeñas poblaciones o etnias que de esta manera consiguen afirmar las propias raíces históricas y la propia cultura.
Muy a menudo podemos aprender de ellas y, con esto, aprender a modificar nuestro comportamiento permeándolo de los significados y de los hábitos mentales sugeridos a través de la filosofía del ejercicio del juego.
Los historiadores piensan que el juego del Go nació en el Tibet unos 2000 años a.C., para difundirse en China mil trescientos años más tarde y después en Japón en 754 d.C. Aquí prosperó hasta el punto que los mejores maestros de este juego son hoy día los japoneses.
En el espíritu del Go ha permanecido algo que recuerda la sensación de inmanencia que inspiran las montañas del Himalaya. Posee una fascinación sutil e indescriptiblemente lúcida que cautiva el ánimo del jugador. En este caso, la sugestión de la filosofía oriental que impregna el Go es profunda, pero no tan alejada para ser incomprensible a nosotros los occidentales.
Fijemos nuestra atención sobre este tema: nos podemos preguntar qué influjo pudo tener el estilo de vida de las poblaciones tibetanas sobre el nacimiento del Go. Dar una respuesta a esta cuestión es cuando menos difícil, aunque podemos intentar conocer mejor aquellas poblaciones con la esperanza de adquirir instrumentos de pensamiento más útiles a esta investigación.
Los habitantes de la altiplanicie tibetana han vivido durante siglos en un absoluto aislamiento, y todavía hoy el Tibet es uno de los lugares más desconocidos de la Tierra.
En una altiplanicie situada entre los 4.000 y los 6.000 metros por encima del nivel del mar, y completamente rodeada de una cadena montañosa con cimas de una altura de más de 8.000 metros, la población seminómada está obligada



La hierática calma de un monasterio tibetano.

a afrontar una existencia durísima por la escasez de llanuras fértiles producida por una tremenda aspereza del clima.

Por tanto no es raro que el Go sea un juego estático, cuya dinámica se expresa preferentemente por una elección de «posiciones»: ¡A aquella altura es difícil correr la maratón!

El Go ha sido objeto de profundos estudios desde 1605, cuando el Shogun Leyasu Tokugawa fundó una academia nacional estable y estableció la conocida figura del maestro de Go.

Hoy hay una masificada expansión del número de cultivadores y jugadores: en Japón se cuentan diez millones de jugadores y, en Estados Unidos, los clubs de Go proliferan en las grandes ciudades y en los campus universitarios. En las librerías universitarias abundan los textos sobre esta materia.

Muchos jugadores pasan del ajedrez al Go seducidos por la simplicidad de sus reglas y al mismo tiempo por la casi inexcrutable complejidad de las combinaciones que permiten.

El jugador está indudablemente estimulado por la curiosidad hacia la simulación, y se hace sensible a la fascinación que ésta ejercita.

El ajedrez simula un torneo entre caballeros medievales, una guerra entre castellanos combatida de una roca a otra, cuyos protagonistas sucumben en cruentos combates atravesados por la lanza del enemigo en singular tensión producida por el continuo movimiento y transformación de los frentes.

La simulación propuesta en el juego del Go tiene una representación diferente y de significado más sutil: en él, la supervivencia y la victoria no se obtienen a través de la aniquilación física del

adversario, sino consiguiendo garantizarse el dominio sobre el número más grande de áreas vitales. La contienda entre las rocas se dilucida aquí en términos de economía política, y tiende a la consecución de una hegemonía estratégica que reduce la fuerza del enemigo, depauperando sus recursos y minando su vitalidad.

La victoria se asigna al final mediante una especie de cálculo burocrático: como si se contasen los campos de grano, los cultivos de arroz, las cabañas de ganado, las fuentes, los pozos y las casas que constituyen la riqueza y el patrimonio de una población.

En el Go no se vive el stress del videojuego: no hay blancos, silbidos ni explosiones; la habilidad manual está reducida al mínimo, la inteligencia está estimulada al máximo.

El Go se juega sobre un tablero cuadrado en el que hay reproducida una rejilla que tiene 19 intersecciones por lado.

En el Go Ban (nombre japonés del tablero), los jugadores depositan 361 fichas, 181 negras y 180 blancas, de acuerdo con los cruces de las líneas sobre el tablero. La tradición exige que las fichas (o peones) negras se obtengan de lastras de pizarra y que, en cambio las blancas se corten en madreperla.

El objetivo del jugador de Go es el de encerrar con las propias fichas el espacio o territorio más amplio posible, cuya superficie se mide contando los cruces libres que han quedado encerrados en él. Los jugadores proceden en este sentido, depositando uno después de otro sus propias fichas en base a la estrategia de juego que prefieran implantar. Inevitablemente, dado que el Go Ban es un espacio finito dentro del cual las jugadas de los contendientes pueden dar lugar a conflictos, se han impuesto reglas precisas de comportamiento que rigen las elecciones de los jugadores.

Las fichas se colocan en un cruce cualquiera del Go Ban que queda libre. La elección es definitiva, en cuanto que la ficha ya no podrá retirarse de la posición en que está, ni podrá desplazarse a otra posición.

En el intento de dominar (encerrándolo) un gran espacio del Go Ban, el jugador puede verse frente a la necesidad de eliminar algunas fichas del adversario: para ello debe seguir el dictado operativo de la REGLA DE CAPTURA, mediante la cual podrá tomar las otras fichas del Go Ban tratándolas como «prisioneras».

Las figuras de la página siguiente muestran algunos ejemplos de áreas encerradas de fichas del mismo color: los puntos marcados con la cruz están en el área respectivamente del Blanco (A, B) y del Negro (C). En estas fases del juego el Blanco tiene cuatro puntos en A (el área en cuestión está completamente rodeada de fichas blancas) y cuatro puntos en B, en un área sólo en parte envuelta por fichas de su color; en el último caso han utilizado la regla que permite emplear los lados exteriores del Go Ban con el fin de prolongar idealmente la cadena de fichas blancas con las «fichas fantasma». Como ya se ha anticipado, un jugador puede aumentar su propia área capturando las fichas del adversario, ya sea una por una o en grupos. También aquí el Go se distingue por un refinamiento: la captura de una o más fichas no se produce por eliminación o salto físico como en las damas. Por el contrario, una ficha termina en el botín del adversario cuando ha quedado rodeada de fichas del color opuesto, o sea cuando ésta ya no tiene «libertad» o, en otras palabras, en el momento en que ya no queda ningún cruce libre «vertical y horizontal» alrededor de la ficha.

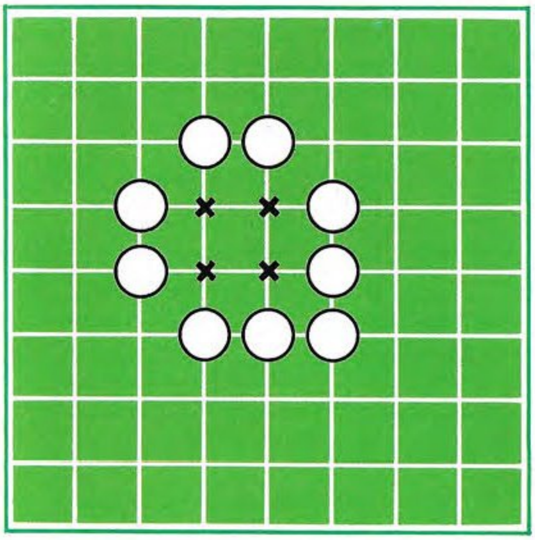
Para poseer un área debe encerrarse con una cadena de fichas, todas rigurosamente del mismo color. De ahí que esta configuración asuma una importancia determinante; se define como GRUPO DE CONEXION.

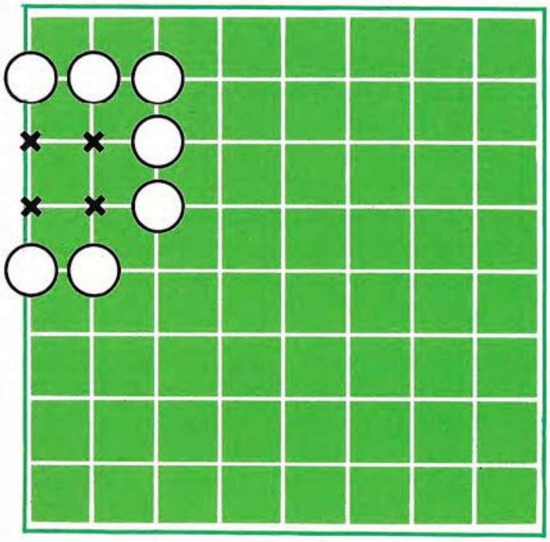
Gran parte de la dinámica del juego y de las escaramuzas entre los dos contendientes sirve a menudo para impedir la conexión de las fichas del adversario en grupos numerosos.

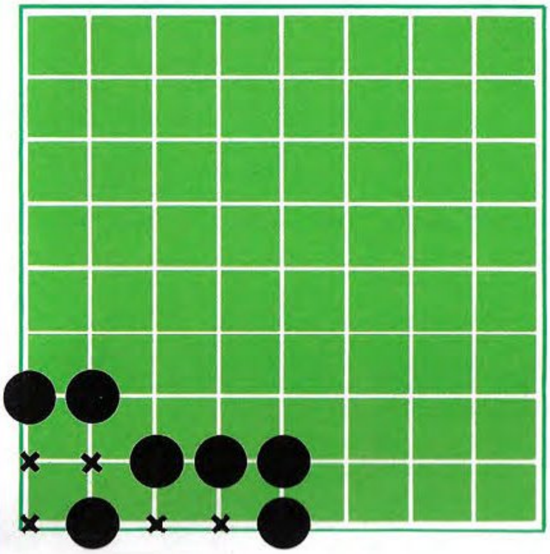
Por tanto, en el Go Ban es útil capturar pequeños espacios todavía no conectados entre sí, recordando sin embargo que el objetivo último del Go es la posesión de territorios y no la captura de fichas: ¡Es inútil eliminar al enemigo si después no se tiene de qué vivir! La captura de un grupo se produce mediante la saturación, con las fichas del adversario privadas de todas sus libertades: en este caso, las fichas se levantan del Go Ban y se guardan como prisioneras hasta el recuento final.

El juego termina cuando los dos contendientes expresan de común acuerdo la convicción de que ya no hay más motivo (ni conquista de territorio, ni captura de prisioneros) para efectuar otras jugadas, que es cuando los dos «pasan» uno después de otro.

Pasar es una decisión lícita en cualquier momento de la partida pero, como es natural, debe tenerse bien presente la ventaja que de este







Las tres imágenes del Go Ban ejemplifican tres modos diferentes de conseguir la posesión de un área libre.

Arriba a la izquierda, el Blanco ha ocupado cuatro posiciones depositando nueve fichas.

Arriba a la derecha se ha ocupado el mismo número de posiciones con 7 fichas, utilizando las «fichas fantasma».

Aquí a la izquierda, el Negro ha ocupado cinco posiciones colocando siete fichas; sin embargo, una de ellas ocupa una posición que podría estar libre.

modo se ofrece al adversario. Normalmente se efectúa esta elección cuando se quiere conceder deliberadamente la captura de un grupo de conexión para el cual se ha perdido cualquier esperanza de salvación y cualquier otra jugada resultaría inútil y dispersiva.

Vencerá el jugador que consiga la puntuación más alta: la puntuación está determinada por el número de los cruces envueltos por los propios grupos, a los cuales se han sustraído las fichas prisioneras en posesión del adversario.

En realidad, las reglas que gobiernan el Go son muchas más y bastante complejas; para no hacer demasiado complicado y difícil el trabajo de nuestro autómata, lo hemos simplificado voluntariamente intentando no falsear el significado y la naturaleza del juego. Por ello, el autómata jugará sobre un Go Ban reducido de 9 × 9 cruces. Las reglas restantes, útiles a los fines de un desarrollo correcto y canónico del Go, se describen en la tabla de la página siguiente.

Para poder construir un adversario artificial, un autómata con el que jugaremos las partidas de Go, debemos analizar un poco cuáles son los criterios que permiten desarrollar correctamente la dinámica del juego.

En este sentido, el lector podrá desarrollar autónomamente una investigación de la cual a conti-

LAS REGLAS DEL GO

1) El Negro empieza el juego colocando, alternativamente con el Blanco, una ficha cada vez sobre el campo de juego.
Si el Negro tiene el derecho de handicap, coloca todas las fichas de handicap al principio de la partida.
Las fichas se colocan sobre los cruces libres del Go Ban.

2) Un jugador puede elegir no efectuar una jugada saltando su propio turno.

3) La victoria se calcula sobre la posesión, por parte de uno de los dos jugadores, del mayor número de cruces libres. La partida con igual número se empata.

4) Un grupo de conexión o una ficha pueden capturarse cuando están privados de libertad.

5) Una ficha prisionera corresponde a un cruce libre menos en la puntuación del jugador que la ha perdido.

6) Está prohibido recrear una situación ya existente en el Go Ban.

7) La colocación de una ficha en un punto privado de libertad se finaliza con la captura de las fichas del adversario.

8) La partida termina por mutuo acuerdo de los jugadores.

9) Al final de la partida se procede al recuento de las fichas «muertas».

10) Un SEKI no da prisioneros ni puntos.

11) Al final de la partida deben cerrarse todos los KO y todos los cruces neutrales.

nuación se indican algunos rudimentos: la descripción de los modelos mentales que gobiernan nuestro comportamiento en una determinada ocasión (ver la del juego) es de por sí un acto al cual debe prestarse la máxima atención. Sobre todo debe tenerse la conciencia precisa de que, al hablar de ellos, se citan interpretaciones fundadas en teorías que, si bien fascinantes, deben siempre afrontarse con un maduro espíritu crítico.
La historia de la literatura es rica en ejemplos en que el autor lleva el sujeto de su obra a una transfiguración hacia un ser dotado de poderes sobrehumanos referidos a veces a las meras dotes físicas y otras veces, en cambio, a atributos intelectuales.
Este ser maravilloso puede realizarse por transformación primero del personaje protagonista (por ejemplo, el tranquilo periodista que se transforma en Superman) o bien tomar vida de sustancias y componentes puestos a disposición de la naturaleza y sintetizados por un genial científico (el clásico ejemplo de Frankenstein).
No creo que entre nosotros existan tranquilos geómetras procedentes del planeta Krypton, ni quiero sugerir la creación de un fantástico jugador de Go de aspecto tétrico y con dos tornillos en el cuello; en este caso deberemos abstenernos de atribuir características antropomórficas a nuestro autómata y esto simplificará el duro trabajo que hemos emprendido.
El pequeño procesador personal que todos nosotros poseemos (o poseeremos) verosímilmente podrá ser el lugar físico en el que instalar este mínimo de (aparente) inteligencia con la cual nos mediremos jugando las partidas de Go.
Este autómata será el producto de la unión de algunos «ladrillos», o procedimientos que, como el barón de Frankenstein, iremos a cavar en el interior de la lógica del juego, afinando así nuestras capacidades lógico-analíticas. Cada uno será después libre de encontrar el lugar a su juicio más conveniente (desde el cómodo salón de su casa hasta el tremebundo castillo de Transilvania) para crear su criatura.
En este punto es útil considerar que el autómata responderá con inteligencia (siempre que sea ésta una característica que se le pueda atribuir) proporcional a la que nosotros mismos conseguiremos profundizar en su interior, utilizando como base el conocimiento que habremos madurado con respecto a la lógica del juego.

(continúa en la pág. 1269)

La utilización de estas funciones necesita un mínimo de atención cuando los programas deben emplearse en diferentes ordenadores. Como se ha dicho anteriormente, el juego de los caracteres disponibles no siempre es el mismo, por lo que, por ejemplo, la función

SUCC ('Z')

podría dar resultados diferentes en calculadores diferentes.
En el tipo de carácter pueden aplicarse los operadores relacionales >, <, =, > =, < = y < >. Estos operadores evidentemente comparan los números correspondientes a la codificación interna de los caracteres, por lo que resultará

'A' < 'B'

porque

ORD ('A') < ORD ('B')

También en el caso de los operadores debe prestarse atención a la correcta utilización en calculadores diferentes.

Tipo booleano. El conjunto de los datos de tipo booleano está compuesto por las dos constantes «verdadero» y «falso». Un dato de tipo booleano siempre se genera como resultado de una operación relacional aplicada a datos de tipo entero, real o de carácter. La relación

A > 5

será verdadera si la variable A contiene un número mayor de 5 y falsa en el caso de que A contenga un número menor o igual a 5.
Los datos de tipo booleano pueden combinarse entre sí por medio de los operadores lógicos AND, OR y NOT. Por ejemplo, la expresión

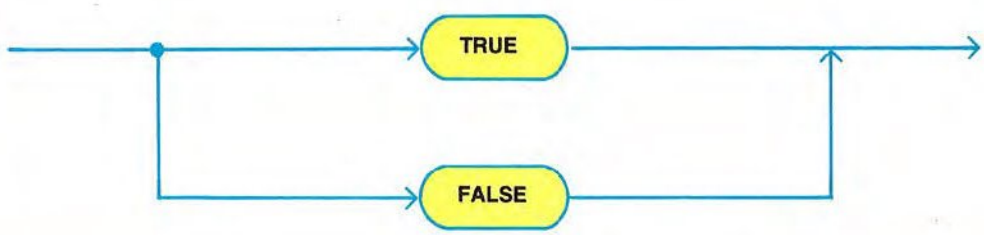
(A = 3) AND (B > 8)

será verdadera si (y sólo si) A resulta igual a 3 y al mismo tiempo B resulta mayor que 8. La sintaxis del dato de tipo booleano se muestra en la figura de esta página.

Advertencia

La forma estándar de escritura de las instrucciones en Pascal prevé el uso de los caracteres alfabéticos en minúscula.
No obstante se ha elegido la presentación de la sintaxis del lenguaje utilizando los caracteres en mayúscula, tanto por motivos de claridad y de homogeneidad con lo tratado anteriormente, como porque en muchas máquinas se emplean exclusivamente los caracteres en mayúscula.
En la mayor parte de los casos, el Compilador Pascal puede aceptar indiferentemente las instrucciones escritas en caracteres en mayúscula o en minúscula.

SINTAXIS DEL TIPO BOOLEANO



Las declaraciones de tipo

Hasta ahora se han analizado los tipos de datos escalares considerados **estándar**, en cuanto ya predefinidos por el Pascal. Antes de describir los otros tipos admitidos analizaremos algunos que en Pascal se definen como **declaraciones**, o sea las definiciones de las constantes y de las variables utilizadas por el programa. Un programa Pascal está subdividido en dos secciones, la primera declarativa y la segunda ejecutiva. La primera sección, a través de la definición de los tipos de datos y de las variables, indica tanto a la segunda sección como al programador cuáles son las estructuras de datos utilizadas.
Es necesario observar que las declaraciones contenidas en la primera sección son coercitivas: el uso en la segunda sección del programa de una variable no declarada en la primera se indica como error. Es decir, el programador está obligado a prever anticipadamente todos los valores que va a utilizar.
Las declaracions pueden ser de dos tipos: de constante o de variable.

La declaración de constante. La declaración de constante es útil cuando se quiere asignar un nombre mnemónico a una constante utilizada frecuentemente en el programa. La sintaxis de la declaración se muestra abajo.
Como se ve, la declaración de constante establece simplemente una equivalencia entre un nombre simbólico y una constante escalar. El valor de dicha constante no podrá cambiarse en el interior del programa; una eventual instrucción que intentase cambiar este valor provocaría una señalización de error por parte del Compilador. Las siguientes declaraciones de constantes son válidas:

```
CONST
        PI = 3.1415927;
        INTERES = 0.11;
        ESPACIO = ' ';
```

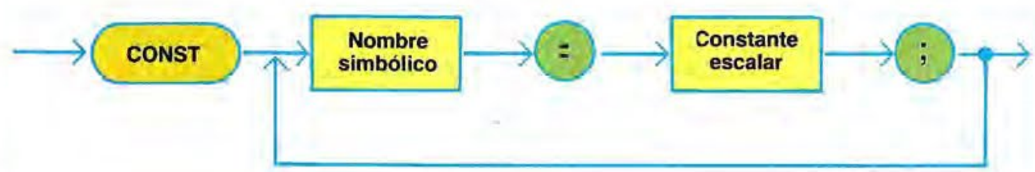
La declaración de constante, además de mejorar la legibilidad del programa, facilita eventuales modificaciones o correcciones.
Consideremos, por ejemplo, un programa para el cálculo de los intereses de una cuenta corriente. Si para la tasa de interés se utiliza un valor numérico en lugar de un nombre simbólico, una eventual variación de la tasa obligaría a volver a buscar en el programa todas las intervenciones de dicha tasa para modificarlas. Utilizando para la constante numérica un nombre simbólico, se consigue el mismo resultado modificando simplemente la declaración de constante.

La declaración de variable. Por variable se entiende una posición de memoria identificada por un nombre simbólico y que puede contener un valor numérico. En algunos lenguajes diferentes del Pascal, definida una primera vez una variable, no existen restricciones sobre el tipo de valor que puede memorizarse en la posición correspondiente. Es decir, a una misma variable se le pueden asignar indiferentemente valores enteros, reales o booleanos. Una de las grandes novedades introducidas por el Pascal es la formalización del concepto de tipo de dato, que conduce a reglas restrictivas en la definición y en la asignación de las variables. Estas reglas tienen tres consecuencias fundamentales:

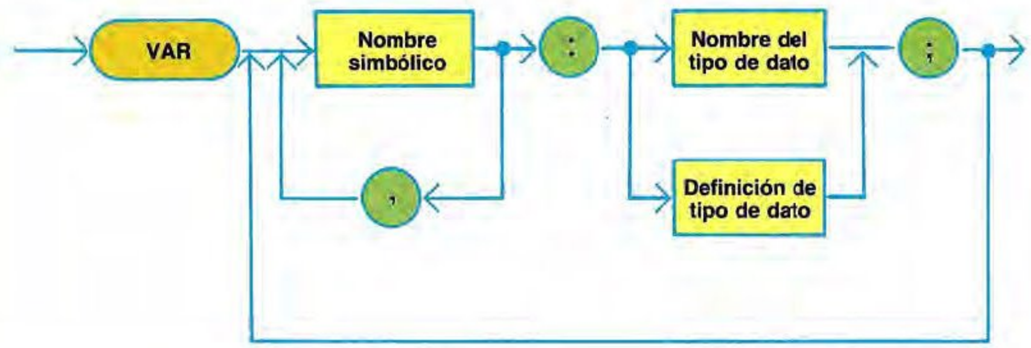
1 / Cada variable escalar puede memorizar valores pertenecientes a un solo tipo de dato

2 / La mayor parte de las funciones previstas

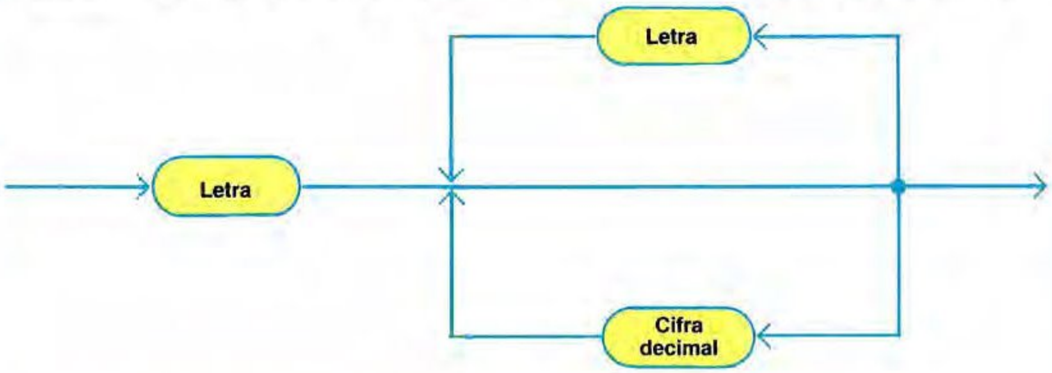
SINTAXIS DE LA DECLARACION DE CONSTANTE



SINTAXIS DE LA DECLARACION DE VARIABLE



CONSTRUCCION DE LOS NOMBRES SIMBOLICOS DE LAS VARIABLES



por el Pascal opera sobre tipos de datos determinantes; se indica un error cada vez que se intenta aplicar dichas funciones a tipos de datos no previstos

3 / Cualquier operación sobre datos de tipo diferente (salvo un solo caso particular) se indica como error.

Estas reglas pueden sembrar pesadas limitaciones en el momento de la escritura de un programa Pascal, pero ofrecen algunas ventajas. Mejoran la legibilidad del programa, puesto que obligan al programador a declarar todas las variables que se utilizarán y el tipo de datos que contendrán. Además, hacen más fácil la búsqueda y la corrección de eventuales errores.
Para crear una variable cualquiera en Pascal es necesario utilizar la declaración VAR, cuya sintaxis se describe en la primera figura de arriba. Como se ve, la declaración debe ir precedida de la palabra clave VAR seguida del nombre o de los nombres que se van a declarar. Las reglas a seguir para la elección del nombre son las mismas vistas para el Basic. El nombre no debe coincidir con una palabra reservada (palabra-instrucción o comando de sistema) y debe empezar con un carácter alfabético; los sucesivos caracteres pueden ser alfabéticos o numéricos. Estas reglas se han resumido en el diagrama la segunda figura de arriba.
Inmediatamente después del nombre (los nombres) debe escribirse el tipo a que pertenece (pertenecen) la variable (las variables). La variable puede ser de tipo estándar, es decir entero,



B Coleman/Marka

Sistema para la gestión computerizada de ficheros.

real, carácter, booleano, o de tipo no estándar, y por tanto definido por el usuario en una declaración precedente de «data type» (el modo para definir tipos de datos no estándar se indicará más adelante).
Ejemplos válidos de declaración son:

```
VAR
   N, I, J     : INTEGER;
   AREA        : REAL;
   VEL, ACC    : REAL;
   A, B, C     : CHAR;
```

Obsérvese que todas las variables utilizadas en un programa Pascal deben declararse obligatoriamente al principio, de otro modo se tendría señalización de error en el momento de la compilación. Algunas variables predefinidas y utilizadas por el sistema son la excepción de esta regla, como por ejemplo la variable MAXINT, la que memoriza el número entero más grande que el ordenador puede procesar.

Tipos de datos escalares no estándar

Además de los tipos de datos estándar descritos, el Pascal también permite utilizar tipos particulares definidos por el usuario en base a las necesidades del problema a resolver. Esta posibilidad resulta particularmente útil cuando los tipos estándar no son adecuados para la resolución de un determinado problema. Utilizando un lenguaje diferente del Pascal se busca normalmente modificar los caracteres del problema para adecuarlos a los tipos de datos disponibles en el lenguaje utilizado. El Pascal supera esta limitación en sentido inverso, permitiendo adaptar los tipos de los datos a las características del problema. Por ejemplo, para escribir un programa que efectúe determinadas acciones para cada día de la semana, podría asociarse a cada día (de lunes a domingo) un número entero de 1 a 7. Esto conduciría a escribir instrucciones condicionales del tipo

```
IF DIA = 6 THEN...
```

que no resultan a primera vista muy legibles, puesto que siempre debe estarse consultando la tabla de equivalencias. En Pascal, el problema puede resolverse definiendo el nuevo tipo de dato DIAS DE LA SEMANA. Las variables asociadas a este nuevo tipo de dato sólo pueden asumir los valores LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMIN-

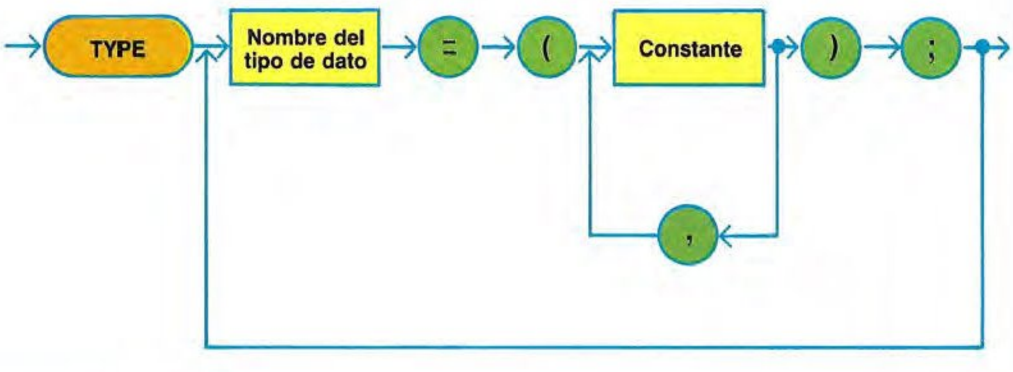
GO. Entonces la instrucción anterior podría escribirse de la siguiente manera:

IF DIA = SABADO THEN...

que seguramente resulta más legible.
La introducción de nuevos tipos de datos definidos por el usuario, además de mejorar la legibilidad del programa, produce otras ventajas.
Los tipos de datos definidos por el usuario pueden definirse de nuevo enumerando singularmente sus elementos (como en el caso de los días de la semana), o como subsistema. En este último caso, el nuevo tipo de dato resulta ser un subsistema (o bien un conjunto más restringido) de un tipo estándar o definido de nuevo.

Tipos de datos definidos de nuevo y enumerados. Como su nombre indica, estos tipos de datos pueden ser creados de nuevo por el usuario, de manera que son completamente independientes de los datos estándar. La creación de nuevos tipos se efectúa con la declaración TYPE, cuya sintaxis se muestra en la figura de abajo. Después de la palabra clave TYPE debe definirse el identificador del tipo, o el nombre asociado al nuevo tipo de dato, al cual deberá hacerse referencia cuando se quiera declarar una variable que pertenezca a este nuevo tipo. El identificador realiza así las mismas funciones de los nombres REAL, INTEGER, CHAR, BOOLEAN, vistas anteriormente. Después del número tipo se relacionan entre paréntesis redondos todas las constantes que pertenecen al nuevo tipo de dato. El orden de relación es importante, puesto que define una relación de ordenación entre las mismas constantes. La relación de ordenación establece que cada constante indicada es «menor» que las que se encuentran a su derecha y «mayor» que aquellas que se encuentran a su izquierda. En la tabla de abajo se indican algunos ejemplos de declaraciones de tipo válidas.

DECLARACION POR TIPOS DE DATOS DEFINIDOS DE NUEVO Y ENUMERADOS



EJEMPLOS DE TIPOS DEFINIDOS DE NUEVO Y ENUMERADOS

```
TYPE
      DIAS DE LA SEMANA = (LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO,
                           DOMINGO);

      MESES = (ENERO, FEBRERO, MARZO, ABRIL, MAYO, JUNIO, JULIO, AGOSTO,
               SEPTIEMBRE, OCTUBRE, NOVIEMBRE DICIEMBRE);

      COLORES = (VIOLETA, AZUL, VERDE, AMARILLO, NARANJA, ROJO);
```

El tipo de dato COLORES está compuesto por las constantes VIOLETA, AZUL, ..., ROJO ordenadas de manera que las siguientes relaciones resultan verdaderas:

VIOLETA < AZUL
NARANJA > VERDE
AMARILLO > VERDE

Aplicar las relaciones comparativas sobre este tipo de datos podría parecer sin sentido. Sin embargo, considerando por ejemplo el tipo de dato MESES, el orden de relación establece unívocamente el ordenado de las constantes; el resultado de

JULIO < AGOSTO

permite utilizar directamente los nombres de los meses para efectuar controles de validez sobre los datos introducidos. Además, establecida la relación de ordenado, pueden aplicarse las dos funciones PRED y SUCC también a los tipos de datos definidos por el usuario. En consecuencia resultará que:

PRED (JULIO) = JUNIO
SUCC (JULIO) = AGOSTO
SUCC (VIERNES) = SABADO
PRED (AMARILLO) = VERDE

Sin embargo, debe prestarse atención a no aplicar la función PRED al primero y la función SUCC al último elemento del tipo.
Además de definir el ordenado en el interior del tipo de dato, la declaración de tipo define unívocamente el conjunto de los valores que puede asumir una variable de este tipo. Esto significa en la práctica que, considerada una variable de nombre TINTA declarada de tipo COLORES, efectuando la asignación

TINTA: = AZUL CELESTE

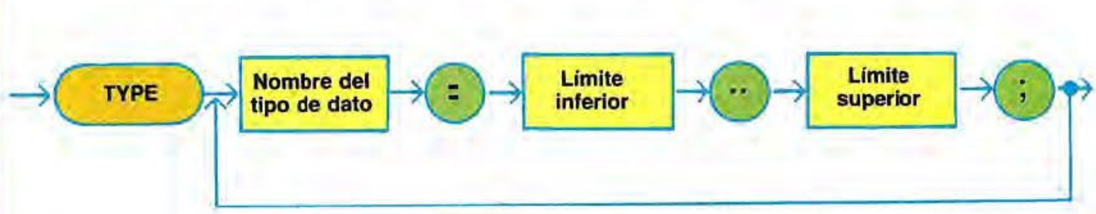
se tendrá la señalización de un error, puesto que la constante AZUL CELESTE no se contempla entre las que forman el tipo de dato COLORES.
De este modo se indican de manera muy sencilla eventuales errores de asignación, y además no es necesario efectuar pesados controles para comprobar la pertenencia de un dato a un determinado tipo.

Tipos de datos subsistema. El tipo de dato subsistema se crea utilizando una porción restringida de los datos que pertenecen a un tipo estándar (excluido el tipo REAL) o bien a un nuevo tipo ya definido por el usuario.
Como se muestra en la figura de abajo, el subsistema se establece definiendo en el interior de un tipo de partida un límite inferior y uno superior, y considerando como pertenecientes al subsistema todos los datos que están comprendidos entre el límite inferior y el superior (extremos incluidos).
Los siguientes ejemplos son declaraciones de tipo subsistema válidas:

```
TYPE
    VOTOS = 0 .. 10;
    CIFRAS = 0 .. 9;
    PRIMAVERA = MARZO .. JUNIO;
    DIAS-LABORABLES = LUNES .. VIERNES.
```

En el primer caso, el subsistema VOTOS está compuesto por números enteros comprendidos entre 0 y 10. En cambio, el tercer y cuarto caso corresponden a tipos definidos por el usuario, por lo que el tipo PRIMAVERA, compuesto por los meses que van de marzo a junio, resultan

DECLARACIONES PARA LOS TIPOS DE DATOS SUBSISTEMA



ser un subsistema del tipo MESES definido anteriormente.
Análogamente a lo que se ha visto para los tipos definidos de nuevo, la utilización de los tipos subsistema permite tanto aumentar la legibilidasd de un programa como efectuar una señalización automática de los errores de asignación. Por ejemplo, declarando una variable de nombre MEDIA de tipo entero, implícitamente se admite que ésta podrá asumir todos los valores posibles para los números enteros. Pero si la misma variable la declaramos del tipo VOTOS (definidos en el ejemplo), entonces, la simple lectura del programa permite comprender que la variable MEDIA podrá asumir, en cada punto del programa, los únicos valores enteros comprendidos entre 0 y 10 (con el significado de media de los votos).

Las primeras instrucciones del Pascal

El conocimiento de los conceptos de tipo de dato, constante, variable y declaración es esencial para escribir programas en Pascal que sean aceptados correctamente por el Compilador. Pero antes de llegar a la escritura de un programa es necesario introducir algunas informaciones que corresponden a la evaluación de las expresiones aritméticas y booleanas, al uso de las funciones y a la asignación de los valores a las variables.

Expresiones aritméticas y booleanas

En la programación en Pascal debe prestarse mucha atención al uso correcto de los tipos de datos, de modo especial en la escritura de las expresiones aritméticas. En efecto, normalmente no se permite insertar en una misma expresión tipos de datos diferentes. La única excepción a esta regla general se tiene en las expresiones aritméticas que necesitan dos números reales para proporcionar un resultado real; en este caso, uno de los dos números reales puede sustituirse por uno entero. Por ejemplo, sean A y C dos variables reales y B una variable entera; la instrucción

C: = A + B

proporciona un resultado correcto en C porque el Compilador convierte el número entero B en un número real antes de calcular la expresión y de asignar el resultado a la variable C. Éste es el único caso en que se admite utilizar en una expresión tipos de datos diferentes. En el ejemplo, el símbolo + podía sustituirse por uno cualquiera de los símbolos de las otras operaciones previstas para los números reales.
Examinemos ahora algunos ejemplos de operaciones no admitidas:

'1' – 1	el valor 1 entre comillas no representa un número, sino un carácter
TRUE+3	no es posible sumar un número entero a un dato de tipo booleano
'A' + 1	en el resultado de la expresión no es cierto el carácter B

Otro ejemplo en que la utilización de un dato diferente al previsto genera un error, se tiene con el uso de los operadores DIV y MOD sobre números reales. Estos operadores efectúan la división entre valores enteros y, por tanto, necesitan operadores enteros. El operador DIV proporciona la parte entera de la división (11 DIV 4 es igual a 2), mientras que el operador MOD proporciona el resto de la división (11 MOD 4 es igual a 3). La operación 11 DIV 4.0 resulta ilegal, puesto que el divisor es visto por el Compilador Pascal como un número real. No debe olvidarse que un número entero y un número real, a pesar de tener el mismo valor, son representados en la memoria de manera diferente. Los números 4 y 4.0 son iguales como valor, pero tienen una representación binaria diferente.
En el Pascal, como en todos los otros lenguajes, existen reglas que condicionan el cálculo de las expresiones. Este lenguaje, como los otros, define jerarquías de preferencia para los operadores; las operaciones de más alta jerarquía se realizan antes que las de jerarquía más baja. Esto es válido siempre que estén ausentes los paréntesis, los cuales pueden definir un orden de preferencia diferente. La jerarquía de preferencia, en orden decreciente de prioridad, es la siguiente:

1 / expresiones en el interior de un paréntesis
2 / operadores *, /, DIV, MOD
3 / operadores de suma y resta (+, –)

Por tanto, en Pascal, el cálculo de una expresión se realiza de esta manera: antes que nada se calculan las expresiones en el interior de los paréntesis; después se realizan las operaciones

de multiplicación y división y, por último, las de suma y resta. Y así, la expresión 7 + A * 4 se evalúa como si estuviese escrita así:

7 + (A * 4)

En el caso en que en una expresión existan dos operadores con el mismo orden de preferencia (no modificado por la presencia de paréntesis), las expresiones se calculan tal como están escritas, procediendo de izquierda a derecha. Por ejemplo, la expresión

A * 12 / B

se calcula como si estuviese escrita así:

(A * 12) / B

Una fórmula del tipo

A + B + (C – D) * E * F / G

se calcula como si estuviese escrita así:

(A + B) + ((((C – D) * E) * F) / G)

Las reglas de homogeneidad de los datos y el orden de preferencia para la evaluación de los resultados también valen para las expresiones booleanas. Obsérvese que los operadores relacionales (<, < =, =, > =, >) pueden utilizarse para comparar datos de tipo entero, real, carácter, booleano o definido por el usuario. Sin embargo, los tipos de datos no pueden mezclarse, en el sentido de que, por ejemplo, no puede compararse un valor real con uno booleano.

2 > = FALSE

es una operación errónea, en cuanto compara dos datos de tipo diferente.
Los operadores lógicos (AND, OR, NOT), en cambio, aceptan el único tipo de dato booleano para formar un resultado de tipo booleano.
También para la evaluación de las expresiones booleanas existe una jerarquía de preferencia expresable, en orden decreciente de prioridad, según la tabla que sigue

1 / expresiones entre paréntesis
2 / operador lógico NOT
3 / operadores /, DIV, MOD, AND
4 / operadores +, –, OR
5 / operadores relacionales (<, < =, =, > =, >)

Esta jerarquía de preferencia es, como se ve, más complicada que la vista para las expresiones aritméticas. Esto depende del hecho de que las expresiones booleanas puedan contener un mayor número de símbolos y de tipos de datos y, además, por el hecho de que en el interior de estas expresiones también puede haber otras expresiones de tipo numérico.
Por ejemplo, la expresión:

A < B – 1

es una expresión booleana válida, porque, siguiendo las reglas enunciadas, se evalúa como si estuviese escrita en el modo

A < (B – 1)

A este respecto debe decirse que el resultado de la expresión escrita anteriormente habría podido calcularse de manera no ambigua, también en ausencia de reglas de preferencia. De ninguna manera puede calcularse como

(A < B) – 1

porque esto requeriría restar el valor entero 1 al valor booleano (verdadero o falso) resultado de la comparación de A y B. En definitiva puede decirse que en las expresiones booleanas, los operadores relacionales deben evaluarse siempre después de haber calculado las expresiones aritméticas.

Utilización de las funciones del Pascal

En la descripción de los tipos de datos ya se ha presentado el conjunto de las funciones estándar implantadas en el Pascal. La utilización de estas funciones necesita la escritura del nombre de la función a utilizar seguida del argumento, o sea del valor que se quiere aplicar a la función, encerrado entre paréntesis. El resultado de la función, o sea el valor de retorno, puede asignarse a una variable o utilizarse en una expresión aritmética. Por ejemplo, podemos escribir:

(SQR(2) * 9) + 6.5

donde el valor 4 (resultado de la elevación a la potencia de 2) se multiplica por 9 y el resultado se suma a 6.5.



G. Hall/Marka

Pantallas de radar en una sala de control del tráfico aéreo.

Por tanto, el resultado de una función puede utilizarse de la misma manera que una constante o una variable. El argumento de una función puede ser también una constante, una variable, una expresión aritmética o incluso el resultado de otra función. Por tanto, podemos escribir indiferentemente

```
SQR (2)
SQR (A)
SQR (7 * A + 12.5)
SQR (SIN (A))
```

obteniendo siempre expresiones válidas (siempre que la variable A se haya definido de tipo compatible con las funciones SQR y SIN).

El cálculo de las expresiones aritméticas que contienen funciones procede según las reglas vistas anteriormente.

En la utilización de las funciones del Pascal debe prestarse atención a los tipos de datos involucrados en el cálculo de las propias funciones. El argumento sobre el que opera cada función debe pertenecer a un tipo bien preciso y el resultado puede pertenecer a un tipo diferente al del argumento. Por ejemplo, la función TRUNC, que elimina las cifras después de la coma de un número real, tiene por argumento un dato de tipo real y proporciona un resultado de tipo entero. Análogamente, la función CHR tiene por argumento un número entero, mientras que como resultado se obtiene un carácter. Para algunas funciones se admite el uso de argumentos de más tipos diferentes, mientras que el resultado siempre es de un solo tipo perfectamente definido. La relación de las funciones estándar disponibles en Pascal, acompañada de una breve descripción de su significado y de los tipos de datos involucrados, se muestra en la tabla de la página siguiente.

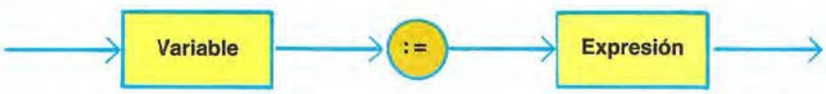
La instrucción de asignación

La instrucción de asignación permite atribuir a una variable un determinado valor. La sintaxis de la instrucción de asignación se muestra en la figura de la página siguiente. Como puede verse, en Pascal se adopta el símbolo : =, definido como **operador de asignación**, diferente del de igualdad, utilizado en todos los otros lenguajes. Esta elección se ha decidido debido a la necesidad de evitar la posible confusión de las evaluaciones de una instrucción del tipo

FUNCIONES ESTANDAR DISPONIBLES EN EL PASCAL

Nombre de la función	Descripción	Tipo de dato argumento	Tipo de dato resultado
ABS	Valor absoluto	Entero o real	Entero
ARCTAN	Arcotangente	Real o entero	Real
CHR	Carácter correspondiente al número	Entero	Carácter
COS	Coseno	Real o entero	Real
EXP	Exponencial $e^{\text{argumento}}$	Real o entero	Real
LN	Logaritmo natural (en base *e*)	Real o entero	Real
ODD	Comprueba si el argumento es impar	Entero	Booleano
ORD	Número correspondiente al carácter	Carácter	Entero
PRED	Dato predecesor del argumento	Cualquier tipo menos el real	Mismo tipo del argumento
ROUND	Redondeado	Real	Entero
SIN	Seno	Real o entero	Real
SQR	Cuadrado del argumento	Entero o real	Mismo tipo del argumento
SQRT	Raíz cuadrada del argumento	Real o entero	Real
SUCC	Dato sucesor del argumento	Cualquier tipo menos el real	Mismo tipo del argumento
TRUNC	Truncado	Real	Entero

SINTAXIS DE LA INSTRUCCION DE ASIGNACION



A = A + 1

que desde el punto de vista matemático no tiene sentido, porque A no puede ser igual a su valor aumentado en uno, mientras que luego lo adquiere si se piensa que el símbolo = no se utiliza como igualdad, sino para indicar que la variable A ahora debe contener el valor anterior incrementado en 1. En el Pascal se ha superado este problema utilizando para las operaciones de asignación y de igualdad dos símbolos diferentes, considerando que estos dos símbolos describen acciones diferentes.

La instrucción de asignación es válida cuando

TEST 21

1 / ¿Cuáles de las siguientes declaraciones de constantes son válidas?

Hallar los errores para las consideradas no válidas.

a) CONST
```
        VALORMAX = 1000;
```
b) CONST
```
        PRIMER : 1;
```
c) CONST
```
        MAXIMO = 500;
        MINIMO = – 50;
```
d) CONST
```
        ALFA = 0 OR 1 OR 2;
```
e) CONST
```
        VALORES = 0 .. 30;
```

2 / Escribir una declaración (única) para las siguientes constantes:

a) el símbolo de la suma (que se llamará PLUS);
b) la constante entera 24, que indica el número de líneas de texto visibles en un monitor (que se llamará MAXLINEAS);
c) los números reales 0 y 100 que representan las temperaturas de congelación y ebullición del agua (que se llamarán respectivamente CONG y EBU).

3 / ¿Cuáles de las siguientes declaraciones son válidas? Para las consideradas no válidas, encontrar los errores.

a) VAR
```
        1CAR, 2CAR, 3CAR : CHAR;
```
b) VAR
```
        CAR1, CAR2, CAR3 : INTEGER;
        IN1              : REAL;
        IN2              : INTEGER;
```
c) VAR
```
        X                : REAL;
        Y                : REAL;
        Z                : REAL;
        XYZ              : CHAR;
```

4 / Escribir una declaración de variable para los siguientes datos, eligiendo oportunamente los nombres de las variables:

a) los colores fundamentales rojo, amarillo, azul;
b) una variable booleana que contenga la respuesta positiva o negativa a una petición;
c) un número entero que tenga la cuenta de las líneas impresas en una página;
d) una variable que contenga el número de matrícula de un automóvil (excluida la indicación de la provincia).

5 / ¿Cuál de las siguientes declaraciones de tipo no son válidas? Para las consideradas no válidas, encontrar los errores.

a) TYPE INTEGER = –MAXINT .. +MAXINT;
b) TYPE GRADOS = 0.0 .. 100.0;
c) TYPE COLORES = (AMARILLO, ROJO, VERDE, MARTILLO, MARRON, AZUL);
d) TYPE PRIMAVERA = MARZO .. JUNIO.

6 / Escribir una declaración (única) para definir los siguientes tipos de datos:

a) los nombres de los palos de las cartas francesas (PALOS);
b) las letras mayúsculas del alfabeto (MAYUSCULAS);
c) el subsistema de los enteros entre –10 y +50 (INTERVALO).

7 / Decir cuáles entre los siguientes valores son datos escalares:

a) el primer elemento de un vector;
b) el nombre 'JUAN PEREZ';
c) el número de las regiones de España;
d) un vector con 20 elementos.

8 / Dado el siguiente tipo

TYPE MESES = (DICIEMBRE, ENERO, FEBRERO, MARZO, JUNIO, JULIO; AGOSTO)

¿cuáles son los resultados de las siguientes funciones?

a) PRED (ENERO);
b) PRED (JUNIO);
c) SUCC (MARZO);
d) SUCC (AGOSTO);
e) PRED (DICIEMBRE).

Las soluciones, en la pág. 1248.

todas las variables que la acompañan se han definido anteriormente, y cuando las de la derecha del operador ya se han asignado. Escribiendo por ejemplo

A:= B + 5.5

debe estarse seguro de que la variable B contenga un valor, especificado por ejemplo por una instrucción anterior de asignación. Además, el tipo de dato obtenido por el cálculo de la expresión debe coincidir con el tipo de dato declarado para la variable A. La única excepción a esta regla, como se ha dicho, es la asignación de un valor entero a una variable de tipo real; en este caso, el número entero se transforma en el número real equivalente y se asigna. En todos los demás casos, los tipos deben coincidir. En el caso de haber introducido las siguientes declaraciones de variable:

```
VAR
      INDICE          : INTEGER;
      CONSONANTE      : CHAR;
      A, B, C         : REAL;
      FLAG            : BOOLEAN;
```

las siguientes instrucciones de asignación resultan válidas:

```
A: = 88.5
B: = 20.
CONSONANTE: = 'K'
INDICE: = 0
FLAG: = A < 50
```

INDICE: = INDICE + 1
C: = B * 6.5
FLAG: = (A < 50) OR (ODD(INDICE))

Las primeras dos asignan dos valores reales a las variables A y B.
Análogamente, las dos siguientes instrucciones asignan valores coherentes con el tipo de dato declarado para las variables. La instrucción

FLAG: = A < 50

asigna el valor FALSE (falso) a la variable booleana FLAG, porque A es mayor que 50. En las dos siguientes instrucciones, la variable INDICE asume un valor igual a 1, mientras que C resulta igual a 130. Finalmente, en la última, la variable booleana FLAG asume el valor TRUE (verdadero), porque a pesar de que A es mayor que 50, la variable entera INDICE es igual a 1 (por tanto impar) y los dos valores booleanos se combinan por medio de la función OR.

Las instrucciones de entrada: READ Y READLN

Las instrucciones que activan en Pascal la función de entrada son READ y READLN; la sintaxis de escritura de las dos se muestra en la figura de esta página. La instrucción READ asigna a las variables indicadas en la «lista de lectura» (enumeradas entre paréntesis) los valores leídos por el dispositivo de entrada. Las variables indicadas en la lista de lectura pueden ser de tipo entero, real, booleano, carácter o subconjunto. La introducción de una variable de cualquier otro tipo generará una señalización de error por parte del Compilador. Haciendo referencia a la declaración del ejemplo examinado anteriormente, los siguientes son ejemplos válidos de instrucciones de entrada:

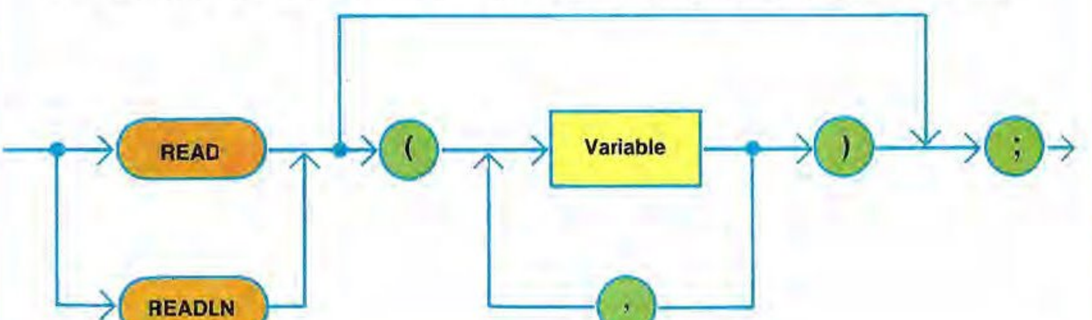
READ (A,B,C);
READ (INDICE,COUNT,FLAG);
READ;

Cuando se ejecuta una instrucción READ deben proporcionarse al calculador los datos necesarios para satisfacerla. El número de los datos a proporcionar debe ser igual al número de las variables contenidas en la lista de lectura, y además, los datos deben coincidir, por tipo y posición, con las correspondientes variables de la lista de lectura. Normalmente, la única excepción admitida es la posibilidad de proporcionar un valor entero a asignar a una variable real. Por ejemplo, habiendo declarado las variables A y B de tipo real y la variable INDICE de tipo entero, se responderá correctamente a la instrucción

READ (A,INDICE,B);

proporcionando los tres números 1.25, 10, 0.
En el caso en que no exista plena correspondencia entre datos y variables, pueden presentarse dos situaciones. En la primera, el programa intenta efectuar una ejecución imposible, por ejemplo un carácter en una variable real, cosa que provoca un error que bloquea la ejecución del programa. La segunda eventualidad, que produce los efectos peores, se tiene cuando el programa asigna un valor erróneo a la variable y continúa la ejecución. En este segundo caso, los efectos deletéreos pueden evitarse insertando oportunamente instrucciones de validación de la entrada que aseguren la correcta asignación de los valores a las variables.
Para completar la descripción de las instruccio-

SINTAXIS DE LAS INSTRUCCIONES DE ENTRADA READ Y READLN



nes de entrada analizaremos la instrucción READLN.
Como se muestra en la figura de la página anterior, la palabra reservada READLN puede escribirse en lugar de la palabra READ conservando la misma sintaxis. La instrucción READLN se utiliza para efectuar la entrada selectiva de datos del teclado o de diversos dispositivos, como por ejemplo el lector de fichas y la unidad de cinta. En estos últimos dispositivos, los datos están organizados normalmente en bloques. Con el uso de la instrucción READLN se extraen algunos datos de un bloque, mientras que los otros se ignoran. Así, si se tienen datos agrupados en bloques de 4, con cada bloque perforado sobre una única ficha (ver) la ejecución de las dos instrucciones de entrada

```
READLN (A,B)
READLN (C,D)
```

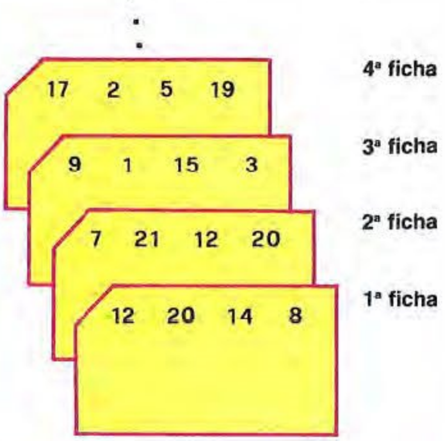
hace que a las variables A y B se asignen los valores

A: = 12
B: = 20

o sea, los dos primeros valores de la primera ficha, mientras que C y D asumirán los valores

C: = 7
D: = 21

FICHAS PERFORADAS QUE CONTIENEN LOS DATOS PARA LA INSTRUCCION READLN



o bien los primeros dos valores presentes en la segunda ficha.

Si se hubiese utilizado en lugar de la instrucción READLN la instrucción sencilla READ, para las cuatro variables se habrían tenido las siguientes asignaciones:

A: = 12
B: = 20
C: = 14
D: = 8

o bien habrían sido asignados todos los valores perforados en la primera ficha.

Las instrucciones de salida: WRITE y WRITELN

Análogamente a lo visto para las instrucciones de entrada, también las instrucciones de salida admiten las dos formas WRITE y WRITELN. La sintaxis de las dos formas se muestra en el diagrama de la página siguiente. El diagrama es bastante complejo y refleja la extrema flexibilidad y potencia de las instrucciones de salida propias del Pascal. Los elementos a imprimir se indican entre paréntesis redondos y pueden presentarse en un número cualquiera. El conjunto de estos elementos se define como «lista de escritura». Cada elemento puede ser una expresión, o incluso una simple variable. Además, el elemento puede ser una cadena de caracteres, que se imprimirá sin las comillas de delimitación. Por ejemplo, suponiendo que a la variable A, declarada de tipo entero, se le haya asignado el valor 4, la instrucción de salida

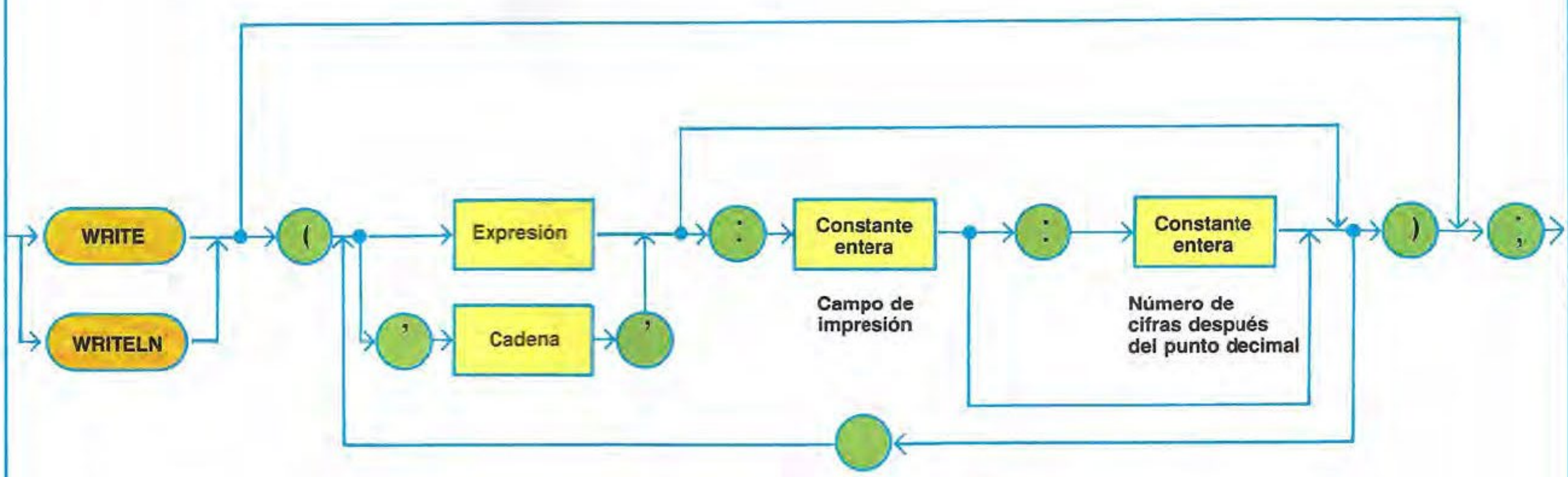
```
WRITE ('A =', A,' El cuadrado de A es ',SQR (A))
```

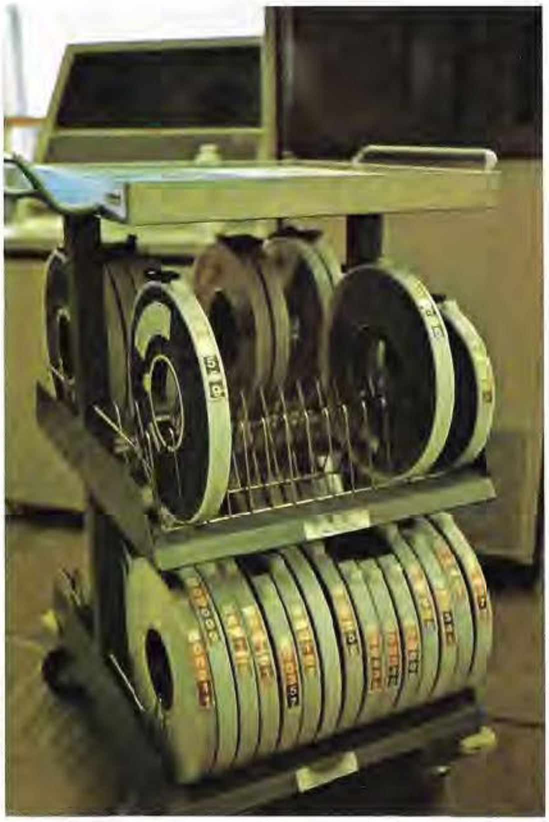
producirá una línea de impresión del tipo

```
A =           4 El cuadrado de A es        16
   10 columnas                      10 cols.
```

Para una variable booleana se imprimirá el valor FALSE o TRUE, mientras que una variable real siempre se imprimirá en notación científica, o sea con un número decimal seguido de una potencia entera de 10, a menos que no se pida explícitamente un formato diferente.
La instrucción WRITE permite efectuar de manera muy simple el formateado de los números y de los caracteres. A cada elemento se asigna en impresión un campo compuesto por un nú-

SINTAXIS DE LAS INSTRUCCIONES DE SALIDA WRITE Y WRITELN





Marka/Infophoto

Carro portacintas.

mero de columnas dependiente del tipo de variable. En el ejemplo se supone que a la variable entera A se le ha asignado un campo de impresión de 10 columnas. Si este número comprende menos de 10 cifras, el ordenador efectúa automáticamente un ajuste de margen del número a la derecha, insertando a su izquierda una cantidad de espacios vacíos suficiente para llenar el campo de impresión previsto. En este caso, como A está compuesto por una sola cifra, el ordenador ha previsto automáticamente insertar a la izquierda del valor nueve espacios en blanco. De ello resulta un ajuste de margen a la derecha del valor.

Las longitudes de los campos de impresión adoptadas por omisión para los diversos tipos de datos son las siguientes:

Enteros = 12 columnas
Reales = 15 columnas (12 cifras significativas con el exponente en la forma E ± xx)

Caracteres = 1 columna
Booleano = 10 columnas
Cadena = Longitud de la cadena.

El Pascal permite también modificar esos valores por omisión escribiendo en la lista de escritura el número de columnas deseado precedido por dos puntos, e inmediatamente después la variable de la que se quiere modificar el campo de impresión. Por ejemplo, la instrucción

WRITE ('A=',A:5,'El cuadrado de A es', SQR(A):5)

produce una línea de impresión del tipo

```
A =          4 El cuadrado de A es        16
   5 columnas                          5 cols.
```

Esta posibilidad de modificar el campo de impresión puede utilizarse para efectuar los encolumnados, de manera análoga a lo que se obtiene en el Basic con la función TAB(X). Por ejemplo, si se quiere imprimir un número de modo que la última cifra quede escrita en la columna 50, basta con introducir la instrucción

WRITE (A:50)

Cuando las variables en impresión son de tipo real, para la instrucción WRITE hay disponible una opción que permite definir el número de las cifras decimales que deben presentarse en impresión. Como se ve en la figura de la página anterior, el número prefijado de cifras después del punto decimal se declara escribiendo inmediatamente después el campo de impresión y separándolo de éste con el símbolo : . Por ejemplo, suponiendo que las variables B y C contienen respectivamente los valores 10.5728 y 152.278, la instrucción

WRITE (B:10:3, C:15)

proporcionará la siguiente impresión:

```
|    10.572|   |1.522780000E + 02|
 10 columnas       15 columnas
```

Además de definir el número de cifras después del punto decimal, el uso de la segunda opción pide explícitamente la impresión de B en forma decimal en coma fija. Cuando esta opción no se pide, las variables reales se imprimen en notación científica. La variable C, por ejemplo, se imprime automáticamente en formato exponencial (en coma flotante). Finalmente, obsérvese que

la última cifra decimal está truncada en lugar de redondeada.
Para terminar la descripción de las instrucciones de salida analizaremos la instrucción WRITELN, cuya sintaxis, descrita en la figura de abajo de la pág. 1241, es del todo análoga a la de la instrucción WRITE. Efectivamente, la única diferencia entre las dos consiste en el hecho de que la WRITELN inserta al final de la línea de impresión el retorno de carro, que hace empezar la siguiente operación de salida en la siguiente línea. Así, por ejemplo, si A y B son dos variables que contienen respectivamente los valores 1 y 2, las dos instrucciones

```
WRITE ('A =', A:4)
WRITE ('B =', B:4)
```

producirán la siguiente línea de impresión:

```
A =   1  B =   2
```

mientras que las instrucciones

```
WRITELN ('A =', A:4)
WRITELN ('B =', B:4)
```

producirán las dos líneas de impresión:

```
A =   1
B =   2
```

La WRITELN, al provocar el avance de una línea de impresión, también puede utilizarse sin parámetros para obtener líneas de espaciado.

Estructura de un programa Pascal

El listado de un programa escrito en Pascal tiene una estructura similar a la indicada en la tabla de abajo, a la que corresponde el diagrama sintáctico de más abajo. En la estructura podemos identificar las dos secciones en que puede dividirse idealmente cada programa.
La primera sección es la cabecera, y consiste en una sola línea que contiene la palabra reser-

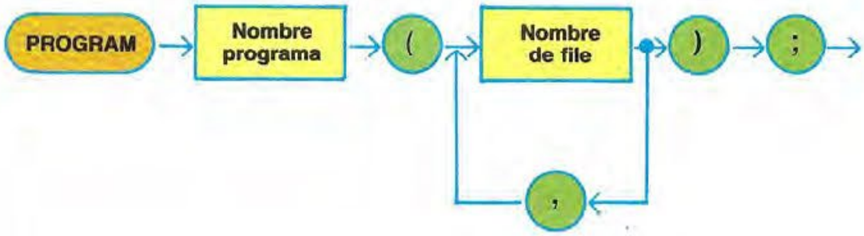
FORMA GENERAL DE UN PROGRAMA PASCAL

```
PROGRAM     nombre (nombre file 1, nombre file 2, ...)
LABEL       declaración... {comentario}
CONST       declaración... {comentario}
TYPE        declaración
VAR         declaración
PROCEDURE   declaración
FUNCTION    declaración
BEGIN
   instrucción; {comentario}
   .......
   .......
   instrucción;
END.
```

DIAGRAMA SINTACTICO QUE REPRESENTA LA ESTRUCTURA DE UN PROGRAMA PASCAL



DIAGRAMA SINTACTICO DE LA CABECERA DEL PROGRAMA



vada PROGRAM seguida del nombre asignado al programa y de la lista de los nombres de los files externos utilizados. La sintaxis de la instrucción PROGRAM se muestra arriba. El nombre del programa no tiene ningún significado para el Sistema Operativo en el programa. En cambio, los files cuyos nombres están contenidos en la lista los utiliza el programador para intercambiar informaciones con el ambiente externo.

Después de la cabecera sigue el cuerpo del programa, compuesto por dos secciones que contienen las declaraciones y las instrucciones. El cuerpo del programa se indica en la figura con el término **block** y también éste puede describirse mediante un programa sintáctico, que se muestra en la página siguiente.

La primera sección del block es la correspondiente a las declaraciones necesarias para describir todos los datos que se utilizarán en el programa. Tres de las seis posibles declaraciones se han analizado anteriormente, o sea las declaraciones CONST, TYPE y VAR. El significado de las declaraciones LABEL, PROCEDURE, FUNCTION se ilustrará a continuación. El orden según el cual deben escribirse las declaraciones es exactamente el indicado en la figura; escribir, por ejemplo, la declaración TYPE antes de la declaración CONST provoca normalmente la señalización de un error.

La segunda sección del block constituye el cuerpo a ejecutar en el programa y se define en inglés con **compound statement**, o sea instrucción compuesta, porque contiene una secuencia de instrucciones Pascal. En la figura de la pág. 1246 se ha representado el esquema sintáctico de la instrucción compuesta. Como puede observarse, todas las instrucciones están encerradas entre las palabras reservadas BEGIN, o sea inicio, y END, o sea final. El significado y la función de estas palabras son sencillos: sirven para encerrar el código a realizar entre dos delimitadores para aumentar su legibilidad aislándolo de la sección de declaración. Por tanto, no se trata de instrucciones propiamente dichas. Sirven exclusivamente para delimitar el conjunto de las instrucciones a ejecutar secuencialmente en el orden en que están escritas. Cada instrucción está separada de la siguiente por el carácter punto y coma (;) o por una palabra reservada. Es decir, el Compilador reconoce el final de una instrucción sólo cuando encuentra el símbolo ; o un conjunto de caracteres que componen una palabra reservada. Normalmente, para evitar errores de compilación, es aconsejable insertar siempre el carácter ; al final de la instrucción, incluso si a veces esto puede resultar superfluo.

Las instrucciones del cuerpo del programa se

DIAGRAMA SINTACTICO DEL BLOCK

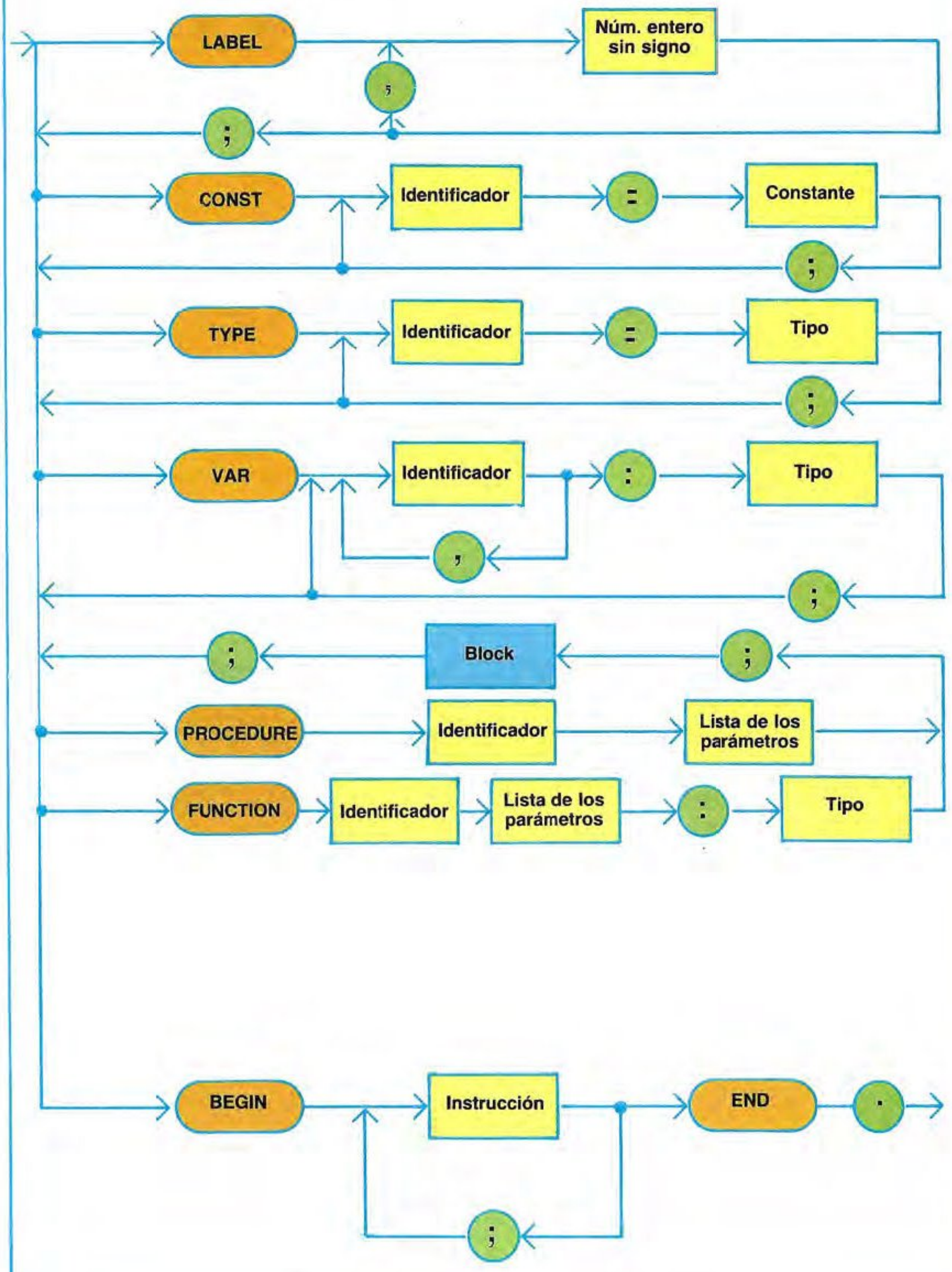
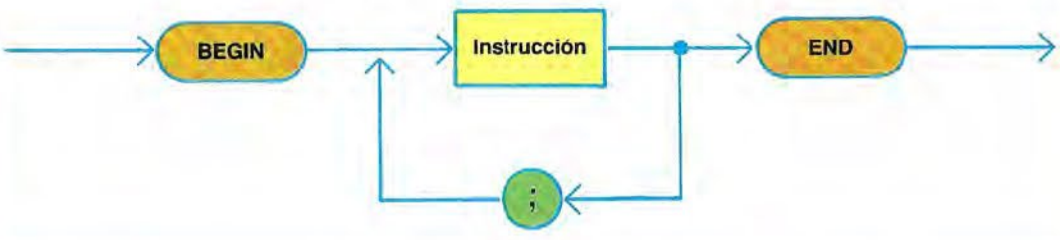


DIAGRAMA SINTACTICO DEL COMPOUND STATEMENT



ejecutan de manera secuencial en el orden en que aparecen. Sin embargo, ciertas instrucciones pueden alterar el flujo normal de control para poder alcanzar el objetivo de un salto condicionado, para repetir más veces un grupo de instrucciones o para mover un procedimiento o función. Después que la última instrucción se ha ejecutado, el programa termina.

Usando las instrucciones Pascal pueden obtenerse del interior del programa las siguientes acciones:

- asignar un valor a una variable (instrucción de asignación)
- reclamar un procedimiento (identificador de procedimiento)
- elegir un conjunto de acciones basándose en ciertos valores (instrucciones IF y CASE)
- repetir un grupo de acciones (instrucciones WHILE, REPEAT, FOR)
- permitir nombrar récords sin proporcionar el nombre (instrucciones WITH)
- transferir el control a otra parte del programa (instrucción GOTO, fuertemente desaconsejada)
- tratar un grupo de instrucciones como una instrucción única (instrucción compuesta)
- no hacer nada (instrucción vacía).

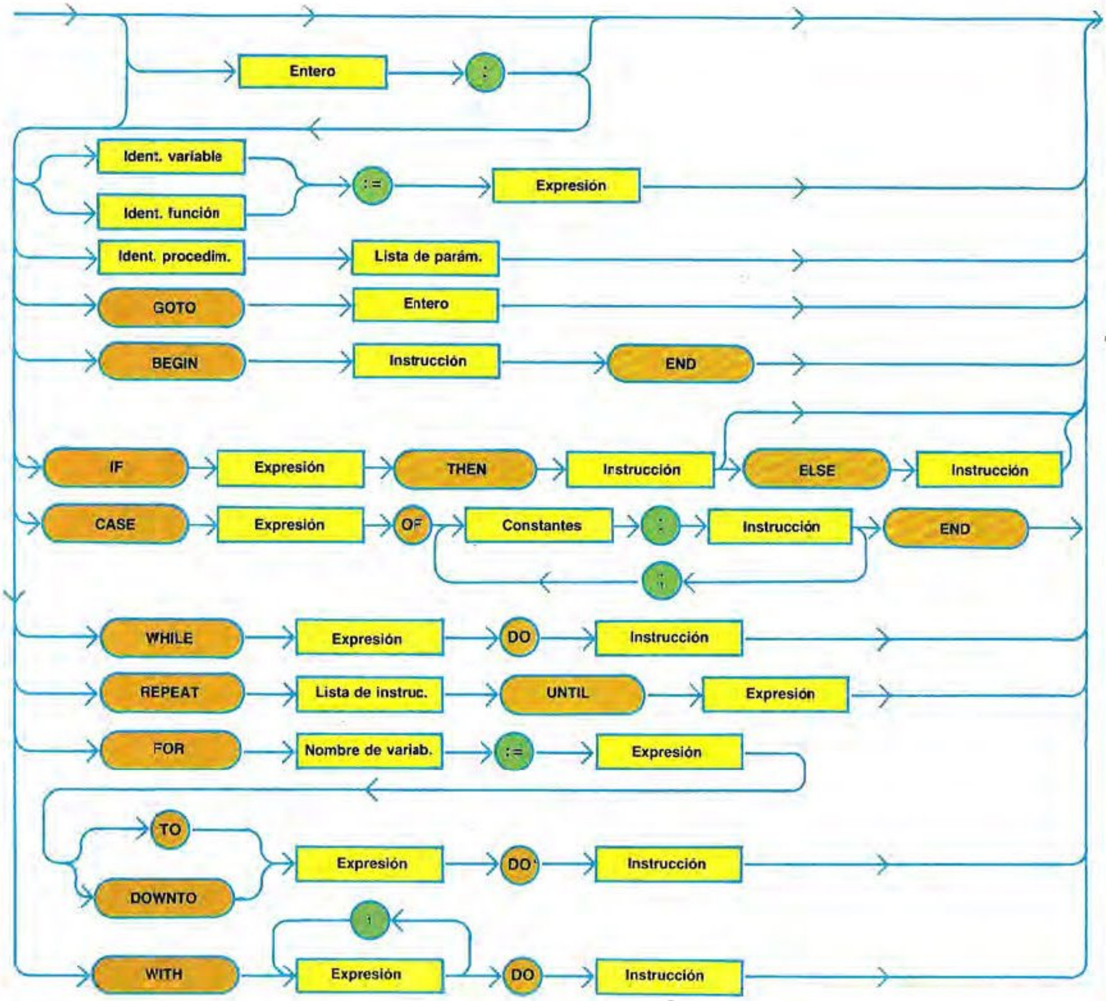
En la figura de la página siguiente se ha ilustrado el diagrama sintáctico completo de las instrucciones Pascal. Algunas instrucciones, como la asignación, la llamada a un procedimiento o el GOTO, se llaman comúnmente **instrucciones simples**. En cambio, las instrucciones IF, CASE, WHILE, REPEAT, FOR y WITH se definen como **instrucciones estructuradas**, puesto que a su vez pueden contener otras instrucciones. En cambio, cada una de las instrucciones estructuradas puede contemplarse como si fuese un subsistema de un bloque. Generalmente, en los Compiladores no existen restricciones sobre el número de las instrucciones estructuradas que pueden anidarse, ni tampoco sobre el número de las instrucciones que forman parte de un bloque.

También en Pascal los programas pueden contener comentarios, que pueden insertarse en cualquier línea y posición delimitadas por los paréntesis cuadrados. A veces, este símbolo no está disponible entre los caracteres del calculador que se emplea, y por este motivo, el Pascal interpreta como comentario también todo lo que va precedido por un par de caracteres *(y terminado por un par de *).

El Pascal permite mucha libertad en la escritura de las instrucciones; pueden insertarse varias sobre una misma línea, y una misma instrucción puede escribirse en varias líneas, siempre que un nombre de variable o un valor numérico no queden a caballo entre dos líneas. Cada instrucción puede escribirse a partir de una columna cualquiera. Esta particularidad puede aprovecharse ventajosamente para efectuar la «indentación» de las instrucciones, o sea el encolumnado de las instrucciones de la manera más adecuada para mejorar la legibilidad de los programas. Un importante elemento que siempre debe existir en un programa Pascal es el punto que indica el fin del programa. Puede imaginarse que el block esté delimitado en la parte superior por la cabecera y por la parte inferior por el símbolo punto. Este acostumbra a ser el aspecto más característico del Pascal. Cada elemento que constituye el programa se ve como una estructura que encierra a otra, la cual, a su vez, está compuesta por otras estructuras, y así sucesivamente.

Esta característica del lenguaje permite definir un problema con la tipología top-down.

DIAGRAMA SINTACTICO DE LAS INSTRUCCIONES PASCAL



Apuntamiento

Palabras reservadas

Expresiones, instrucciones y nombres dados por el programador

SOLUCIONES DEL TEST 21

1 / a) válida;
b) no válida; el símbolo : debe sustituirse por el símbolo = ;
c) válida;
d) no válida; no se admite escribir funciones booleanas en una decalaración de tipo;
e) no válida; no puede definirse una constante con el tipo subconjunto.

2 / CONST
PLUS = '+';
MAXLINEAS = 24;
CONG = 0.;
EBU = 100.;

3 / a) no válida; el primer carácter del nombre de una variable debe ser obligatoriamente una letra;
b) válida;
c) válida.

4 / VAR
a) COLORES: (ROJO, AMARILLO, AZUL);
b) RESPUESTA: BOOLEAN;
c) LINEAS: INTEGER;
d) MATRICULA: INTEGER;

5 / a) válida; se trata de una declaración de tipo predefinida en el Pascal;
b) no válida; no se admite declarar un subsistema del tipo real;
c) válida;
d) no es válida si antes no se ha declarado un tipo (por ejemplo MESES) que contenga los extremos indicados en la declaración.

6 / TYPE
PALOS = (CORAZONES, DIAMANTES, TREBOLES, PICAS);
MAYUSCULAS = 'A' .. 'Z';
INTERVALO = – 10 .. 50;

7 / a) escalar: se trata de un solo número;
b) no es escalar: es una cadena y un dato estructurado.
c) escalar: es un número entero;
d) no escalar: es uno de los dos valores admitidos para el tipo escalar estándar booleano.

8 / a) PRED (ENERO) = DICIEMBRE;
b) PRED (JUNIO) = MARZO;
c) SUCC (MARZO) = JUNIO;
d) SUCC (AGOSTO) = no está definido;
e) PRED (DICIEMBRE) = no está definido.

Las instrucciones de control

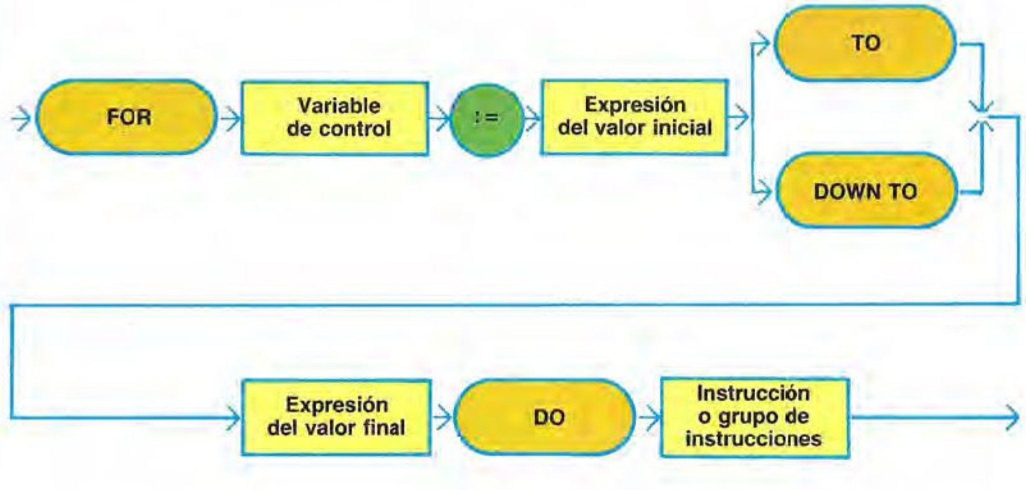
Utilizando las instrucciones descritas hasta ahora pueden escribirse sencillos programas en Pascal que tienen una estructura «secuencial», en los cuales no existe la posibilidad de modificar el flujo de ejecución de las instrucciones según se verifiquen o no ciertas condiciones.
La posibilidad de modificar la secuencia de ejecución de las instrucciones de un programa existe también en el lenguaje Pascal, y se realiza mediante las intrucciones de control. Estas instrucciones son muy importantes, puesto que son necesarias para la resolución por el calculador de los problemas más complejos. El Pascal prevé un conjunto de instrucciones de iteración y de salto (condicionado o no) seguramente más extenso y flexible que los otros lenguajes de uso común como el Basic y el Fortran.

Las instrucciones de iteración

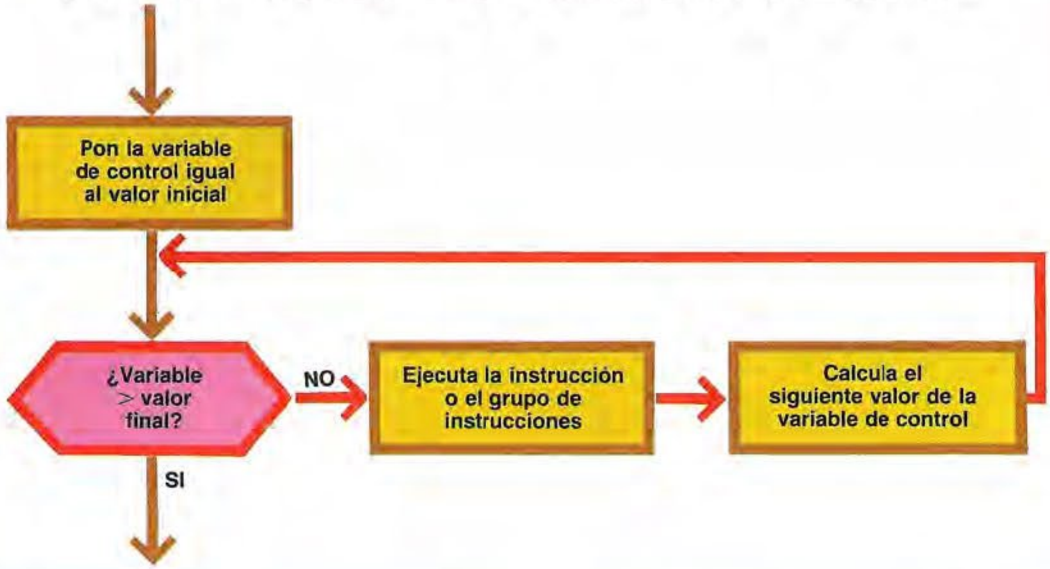
Las instrucciones de iteración permiten la ejecución del mismo grupo de instrucciones un número de veces prefijado o bien dependiente de la verificación de una condición. Además de la instrucción FOR... TO... DO..., presente con algunas diferencias también en Basic, el Pascal admite el uso de otras dos instrucciones, la WHILE... DO... y la REPEAT... UNTIL... Se trata de estructuras muy potentes que permiten resolver de manera rápida todos los problemas en los que se necesita la ejecución repetica de un grupo de instrucciones.

La instrucción FOR... TO.... DO... Se utiliza para ejecutar una instrucción o un grupo de intrucciones todas las veces necesarias para que una variable de control, incrementada en cada iteración, pase de un valor inicial a un valor final prefijados. El diagrama sintáctico de la instrucción se muestra aquí abajo, mientras que en la página siguiente puede verse un diagrama de flujo que ilustra el funcionamiento de la instrucción desde el punto de vista lógico. Esta instrucción Pascal es muy similar a la correspondiente estructura Basic FOR... TO... NEXT..., aunque presenta respecto a ésta algunas pequeñas pero sensibles diferencias. Del diagrama de la página siguiente aparece claro que la instrucción o las instrucciones que constituyen el cuerpo del ciclo nunca se ejecutan si de la primera comparación resulta que la variable de control es mayor que el valor final. En cambio, en el Basic, el cuerpo del ciclo se ejecuta al menos una vez, independientemente del valor inicial de la variable de control. Además, mientras que en el Basic es posible especificar con la palabra reservada STEP el valor de incremento de la variable de control (paso), esto no es posible en Pascal.

DIAGRAMA SINTACTICO DE LA INSTRUCCION FOR... TO... DO...



FUNCIONAMIENTO LOGICO DE LA INSTRUCCION FOR... TO... DO...



Volviendo al diagrama de antes, observamos que los valores iniciales y generales de la variable de control no deben ser necesariamente constantes numéricas, sino que pueden ser también valores de datos de expresiones, evidentemente de tipo congruente con el tipo de dato a que pertenece la variable de control. En ese aspecto, es importante observar que el cálculo de las expresiones que definen los valores inicial y final se realiza sólo una vez, cuando el Compilador encuentra la instrucción FOR... Esto significa que, una vez evaluados el valor inicial y el final, no pueden modificarse durante la ejecución del ciclo. La secuencia de instrucciones

```
K: = 5;
FOR I: = K – 1 TO K + 1 DO
BEGIN
        K: = K – 1;
        WRITELN ('I = ', I:2, 'K =', K:2);
END;
```

produce las siguientes líneas de impresión

```
I = 4   K = 4
I = 5   K = 3
I = 6   K = 2
```

El ciclo se realiza para I que va de 4 a 6, o sea del valor inicial al valor final calculados en base a K = 5; ningún cambio de la variable K en el interior del ciclo puede modificar estos valores.

La variable de control se incremente al siguiente valor cuando en la instrucción se encuenta la palabra reservada TO, mientras que es decrementada si se utiliza la palabra reservada DOWNTO. En este último caso, el valor final debe ser menor que el valor inicial.

El siguiente valor de la variable de control se calcula aplicando la función SUCC en el caso de incremento y la función PRED para el el decremento. Esto significa que el tipo real no puede utilizarse como variable de control, puesto que no se admite como argumento para las funciones SUCC y PRED. Todos los demás tipos de datos sencillos se admiten, por lo que, por ejemplo, es lícito escribir una instrucción del tipo

```
FORD C: = 'A' TO 'Z' DO
```

que en cambio no es aceptada por el intérprete Basic.

El cuerpo del ciclo puede ser dado por una instrucción simple o por una instrucción compuesta, o sea por un conjunto de instrucciones delimitado por las palabras reservadas BEGIN y END. En el interior del cuerpo del ciclo no se admiten instrucciones de asignación de la variable de control, o sea no se admite la modificación de los valores asumidos poco a poco por esta última. Sin embargo es posible asignar su valor a otra variable, que puede utilizarse después libremente.

Dada la secuencia de instrucciones

```
SUMA: = 0;
FOR K: = 1 TO 5 DO
BEGIN
        SUMA: = SUMA + K;
        K: = K + 2;
END;
```

la instrucción

```
K: = K + 2
```

no se admite en el interior del ciclo, y su presencia se señala como error en el momento de la compilación.

Terminada la ejecución de la instrucción FOR... TO... DO..., o bien una vez ejecutado el cuerpo del ciclo con el número de veces prefijado, el valor de la variable de control se hace indefinido y, por tanto, no puede utilizarse en las instrucciones sucesivas, a menos que no se reasigne. El empleo de la instrucción FOR... TO... DO... es particularmente conveniente cuando deben ejecutarse grupos de instrucciones bajo el control de un contador un número de veces conocido a priori. Sin embargo, a veces el número de las repeticiones del ciclo no se sabe a priori, sino que depende de la verificación de ciertas condiciones internas del programa. En este caso, el uso de la instrucción FOR... TO... DO... ya no es conveniente. El Pascal pone a disposición otras dos funciones, la WHILE... DO... y la REPEAT... UNTIL..., que son más flexibles y adecuadas a las situaciones descritas.

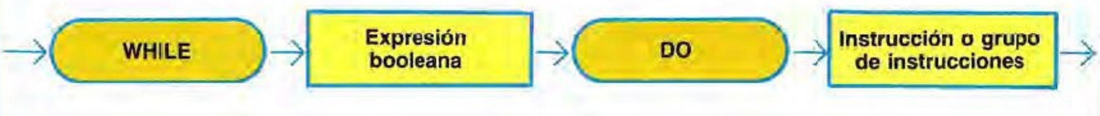
La instrucción WHILE... DO... Esta instrucción hace que un ciclo continúe realizándose hasta que resulte verdadera una determinada condición booleana. La sintaxis se muestra en la figura de abajo. La expresión de control se evalúa al principio y, si resulta verdadera, se ejecuta todo lo que sigue a la palabra reservada DO, o el cuerpo del ciclo. A continuación se recalcula la expresión booleana, y si todavía resulta verdadera, se vuelve a ejecutar el cuerpo del ciclo, y así sucesivamente. El ciclo ya no se repite cuando la expresión se hace falsa. Este modo de trabajar se muestra en el diagrama de flujo de la parte superior de la página siguiente. En el programa de la página siguiente se presenta un ejemplo de utilización de la instrucción WHILE... DO..., que permite imprimir los primeros cien números enteros.

Es importante observar que, a diferencia de todo lo que se ha dicho a propósito de la instrucción FOR... TO... DO..., en el interior del cuerpo del ciclo no sólo se admite modificar la variable de control, sino que esto a veces es necesario para evitar errores. Efectivamente, si en el interior del ciclo no hubiese ninguna instrucción de modificación de la variable de control, y si la expresión booleana resultase verdadera en el momento de la ejecución de la instrucción WHILE... DO..., el ciclo se repetiría indefinidamente. Por tanto, debe prestarse mucha atención al escribir la expresión booleana y al verificar qué variable asume el valor FALSE en el momento en que debe terminar el ciclo. A título de ejemplo, considérese la secuencia de instrucciones escrita de la siguiente manera:

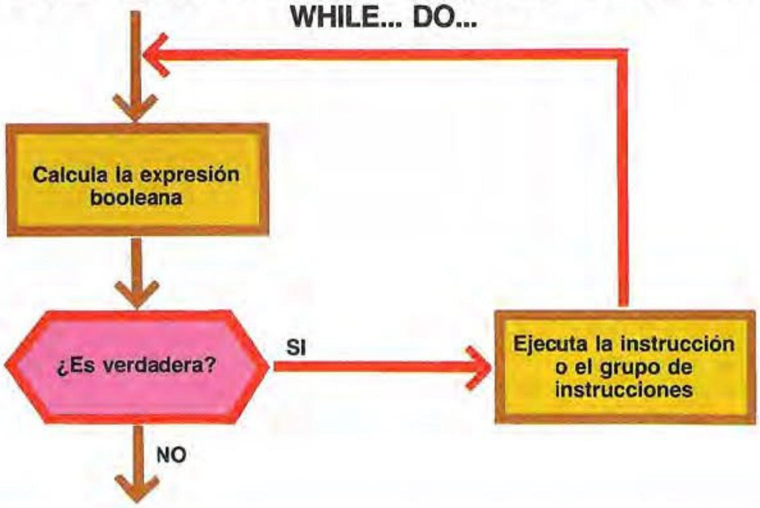
```
READ (INDICE);
WHILE INDICE < >0 DO
  BEGIN
    ......
    INDICE: = INDICE – 1;
    ......
  END;
```

Si a la petición de entrada de la variable INDICE

DIAGRAMA SINTACTICO DE LA INSTRUCCION WHILE... DO...



ESQUEMA LOGICO DE FUNCIONAMIENTO DE LA INSTRUCCION WHILE... DO...



IMPRESION DE LOS PRIMEROS CIEN NUMEROS ENTEROS

```
PROGRAM IMPRIME (INPUT, OUTPUT);

INDICE  INTEGER; (* ES LA VARIABLE QUE SE IMPRIMIRA
   VAR                  Y UTILIZARA PARA EL CONTROL DE FIN DE CICLO *)
BEGIN
        WHILE INDICE < = 100 DO
        BEGIN
                WRITELN (INDICE:5);
                INDICE: = INDICE + 1;
        END;    (* FIN DEL CUERPO DEL CICLO *)
END.            (* FIN DEL PROGRAMA *)
```

se responde con un número negativo, el ciclo se realizará infinitas veces; efectivamente, por ser INDICE decrementado en cada paso, su valor nunca podrá hacerse igual a 0 y la expresión booleana INDICE < >0 resultará siempre verdadera.

El cuerpo del ciclo incluido en una WHILE... DO... podría no ser nunca ejecutado si la expresión booleana resultase falsa en el momento de la primera ejecución de la instrucción. Si sucede que el ciclo se ha ejecutado al menos una vez, prescindiendo del valor inicial de las expresiones booleanas, debe utilizarse la instrucción REPEAT... UNTIL...

La instrucción REPEAT... UNTIL... El diagrama sintáctico de la instrucción se muestra en la parte superior de la página siguiente. Tiene un comportamiento similar al de la WHILE... DO...: un ciclo se repite continuamente hasta que se cumple una determinada condición. Por lo menos hay dos diferencias importantes. La primera consiste en el hecho de que mientras la instrucción WHILE... DO... realiza el test sobre la condición **antes** de ejecutar el cuerpo del ciclo, la REPEAT... UNTIL... realiza el test **después** de haber ejecutado el ciclo. La segunda diferencia es que en la REPEAT... UNTIL..., el cuerpo del ciclo se ejecuta en tanto que la condición de control resulta falsa, mientras que en la instrucción WHILE... DO..., el cuerpo del ciclo se ejecuta mientras la expresión booleana resulta verdadera (la condición de expresión falsa hace terminar dicho ciclo). Esta diferencia de comportamiento puede evaluarse claramente comparando el esquema de flujo lógico de la instrucción

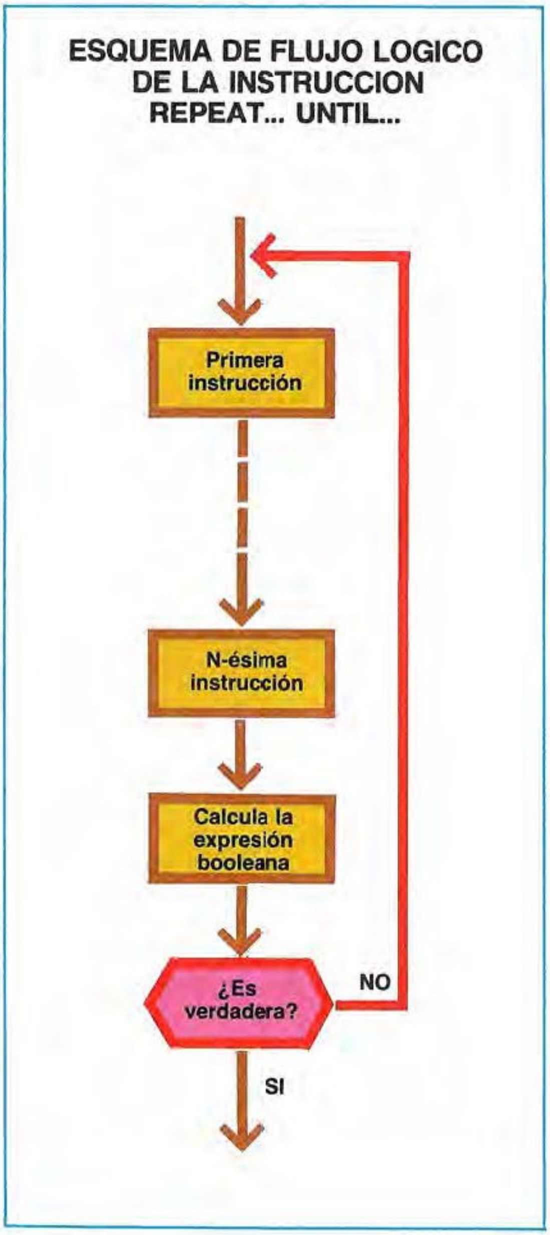


REPEAT... UNTIL..., mostrado en la página siguiente, con el análogo de la WHILE... DO... Del esquema del flujo se deduce que también a diferencia de lo que sucede para la instrucción WHILE... DO..., en la REPEAT... UNTIL... las instrucciones que constituyen el cuerpo del ciclo se ejecutan al menos una vez.
A título de ejemplo, en la parte del programa

```
READ (CONTROL);
REPEAT
        INSTRUCCION 1;
        INSTRUCCION 2;
UNTIL (CONTROL = 0);
```

respondiendo a la instrucción de entrada con el número 0, las instrucciones 1 y 2 se realizan una vez después de terminar el ciclo, porque la condición de control es verdadera.
En la instrucción REPEAT... UNTIL..., si el cuerpo del ciclo está compuesto por más de una instrucción, no es necesario encerrarlo entre los delimitadores BEGIN y END como en la WHILE... DO..., porque las dos palabras reservadas REPEAT y UNTIL tienen las mismas funciones de delimitadores. Naturalmente siempre es posible encerrar el cuerpo del ciclo entre las palabras BEGIN y END, como en el ejemplo

```
REPEAT
 BEGIN
       INSTRUCCION 1;
       INSTRUCCION 2;
       .
       .
       .
       .
 END;
UNTIL (CONTROL = 0);
```



Este modo de escribir no se señala como error, aunque es redundante.

Las instrucciones condicionales

Las instrucciones condicionales son necesarias para mofificar el flujo de ejecución de un programa, o para hacer ejecutar un grupo de instrucciones como alternativa a otras al cumplirse una determinada condición. Además de la instrucción IF... THEN..., presente también en el Basic, en Pascal se admite la instrucción CASE, que permite elegir entre más de dos alternativas.

La instrucción IF... THEN... ELSE... El diagrama sintáctico de la instrucción IF... THEN... se muestra aquí abajo. Su significado es suficientemente claro: la expresión booleana se evalúa, y si resulta verdadera la instrucción que sigue a la palabra reservada THEN, se envía a ejecución. De otro modo, el control del programa pasa a la siguiente instrucción.
Normalmente, la instrucción a realizar puede ser una instrucción compuesta, o sea un conjunto de instrucciones delimitado por las palabras BEGIN y END. Por ejemplo, es posible escribir

```
IF CONTROL >0 THEN A: = + 1;
```

o bien

```
IF CONTROL = 0 THEN
  BEGIN
     A: = A + 1;
     WRITELN ('A =', A:5);
  END;
```

En este segundo caso, si la condición no se cumple, se ejecuta la instrucción que sigue a la palabra clave END. Después de la palabra reservada THEN, puede insertarse una instrucción Pascal cualquiera. Por ejemplo, pueden escribirse instrucciones del tipo

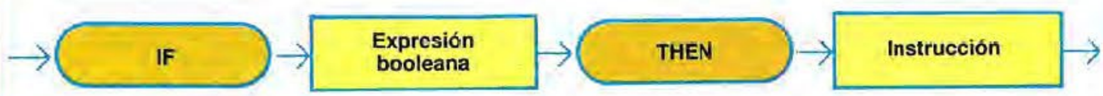
```
IF CONTROL = 0 THEN IF A = 10 THEN...
```

Evidentemente, la segunda instrucción IF... THEN... sólo se ejecutará si se cumple la condición en la variable CONTROL.
La expresión booleana en la que se realiza la verificación también puede construirse utilizando los operadores lógicos AND, NOT, OR. Este aspecto es muy importante, porque permite construir condiciones complejas y que pueden satisfacer exigencias particulares.
Por ejemplo, es posible hacer ejecutar una cierta instrucción si la variable de control está comprendida en un determinado intervalo, o si resulta mayor o menor que un cierto valor.
La línea de programa

```
IF (A> 0) AND (A< = 100) THEN A: = 1
```

pone a 1 la variable A si su valor está comprendido en el intervalo de 0 a 100. El mismo resultado también podría obtenerse concatenando más instrucciones IF... THEN...
La posibilidad de construir condiciones de verificación complejas es muy útil cuando se quiere efectuar la validación de datos en la entrada. Por ejemplo, si se quiere introducir una fecha en el calculador, deberá verificarse que ésta tenga un sentido, de manera que no sea posible introducir valores erróneos.
Este control puede obtenerse con la combinación de las condiciones a verificar; si la fecha proporcionada por el operador no es lícita, podrá imprimirse un mensaje informativo previsto

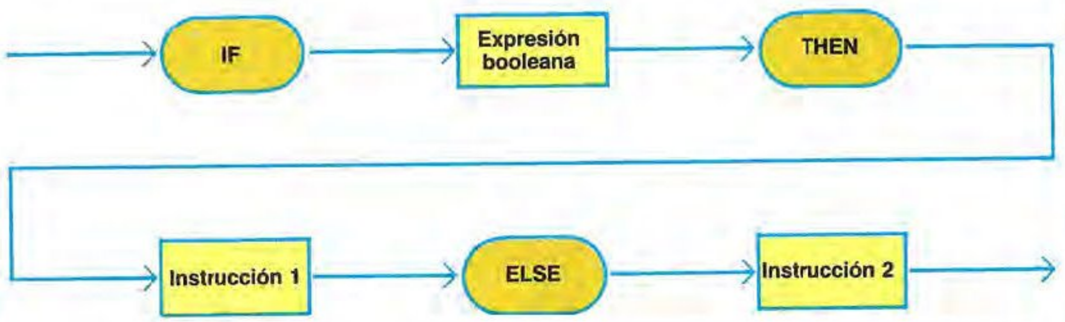
SINTAXIS DE LA INSTRUCCION IF... THEN...



INTRODUCCION Y CONTROL DE UNA FECHA

```
WRITELN ('INTRODUCIR FECHA DE NACIMIENTO');
READLN (DIA, MES, AÑO);
IF (DIA > 31) OR (MES > 12) THEN
  BEGIN
        WRITELN ('LA FECHA NO ES CORRECTA; REPETIR INTRODUCCION');
        READLN (DIA, MES, AÑO);
  END;
........
```

SINTAXIS DE LA INSTRUCCION IF... THEN... ELSE...



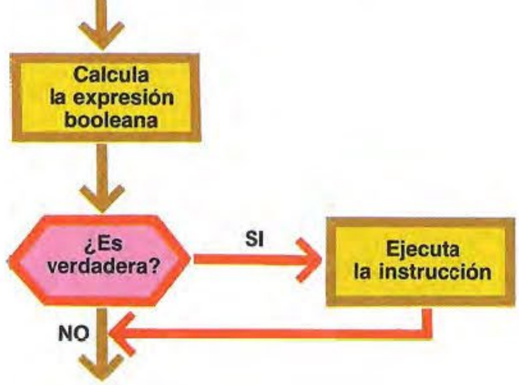
oportunamente y el programa repetirá la petición de introducción. La parte de programa de la parte superior de esta página es un ejemplo. La primera instrucción sirve para advertir al operador que deberá introducir una fecha de nacimiento, y con la segunda instrucción, los tres números que la componen son leídos y asignados a las tres variables DIA, MES y AÑO. A continuación estos valores se controlan: si DIA resulta mayor que 31 o bien MES mayor que 12 (operador lógico OR), entonces se imprime un mensaje que advierte del error e invita a repetir la introducción; en ausencia de errores, el programa puede proceder en secuencia. El ejemplo es evidentemente del todo indicativo, porque también debería controlarse que los números introducidos no sean menores o iguales a 0, o que el día del mes no vaya más allá del campo de los valores permitidos por cada mes. Utilizando la instrucción IF... THEN... puede hacerse que una cierta instrucción se ejecute o no en función de que se cumpla la condición de control. En cambio, a veces es necesario ejecutar, según el resultado del test de control, una u otra de dos intrucciones (o grupos de instrucciones). En Pascal esto puede obtenerse con la instrucción IF... THEN... ELSE, cuyo diagrama sintáctico se muestra en la figura de arriba.

La ejecución de esta instrucción se inicia primeramente con la evaluación de la expresión booleana; si ésta resulta verdadera se realiza la instrucción 1; en cambio, si resulta falsa, se realiza la instrucción 2.

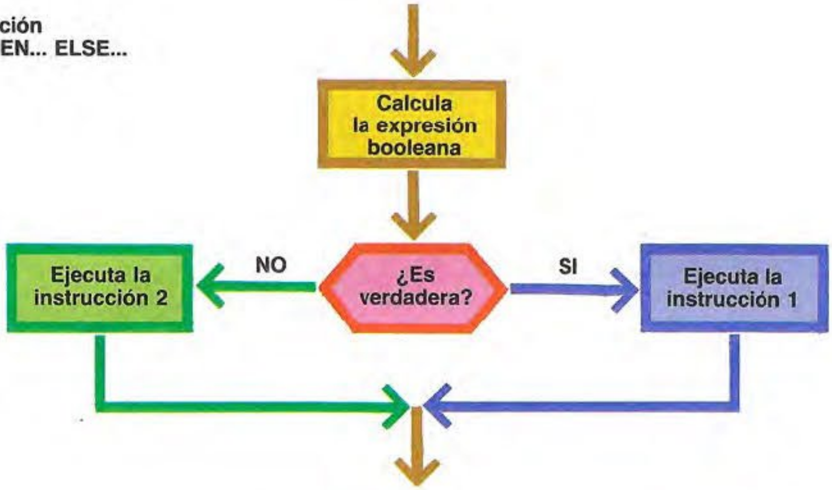
En la figura de la página siguiente se muestra la diferencia existente entre la instrucción examinada y la instrucción IF... THEN... Volviendo al ejemplo presentado anteriormente, relativo a la validación de las fechas de nacimiento, esta nueva instrucción puede utilizarse para escribir una sección de programa que, además de advertir al operador si la fecha es errónea, presente en el monitor la fecha introducida si ésta resulta correcta. En la página siguiente puede verse un ejemplo.

ESQUEMAS LOGICOS DE LAS INSTRUCCIONES IF... THEN... E IF... THEN... ELSE... COMPARADOS

Instrucción IF... THEN...



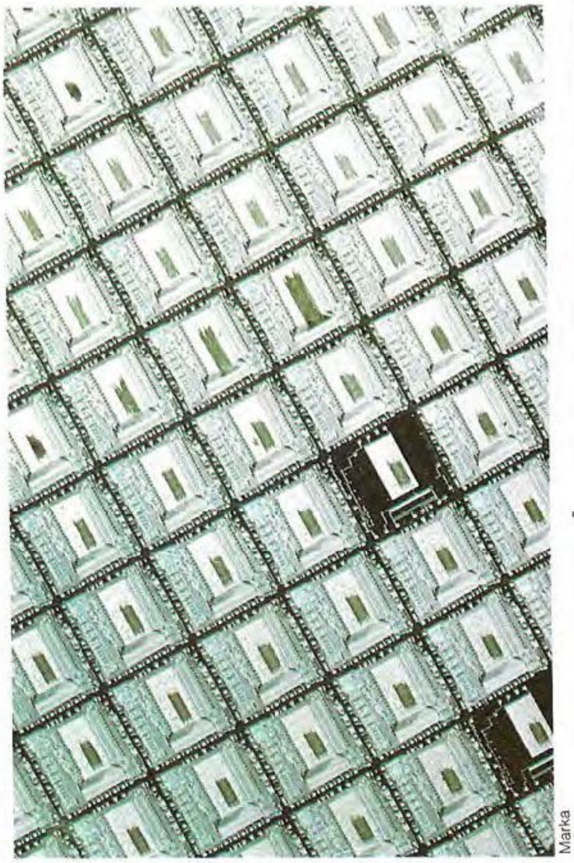
Instrucción IF... THEN... ELSE...



APLICACION DE LA INSTRUCCION IF... THEN... ELSE...

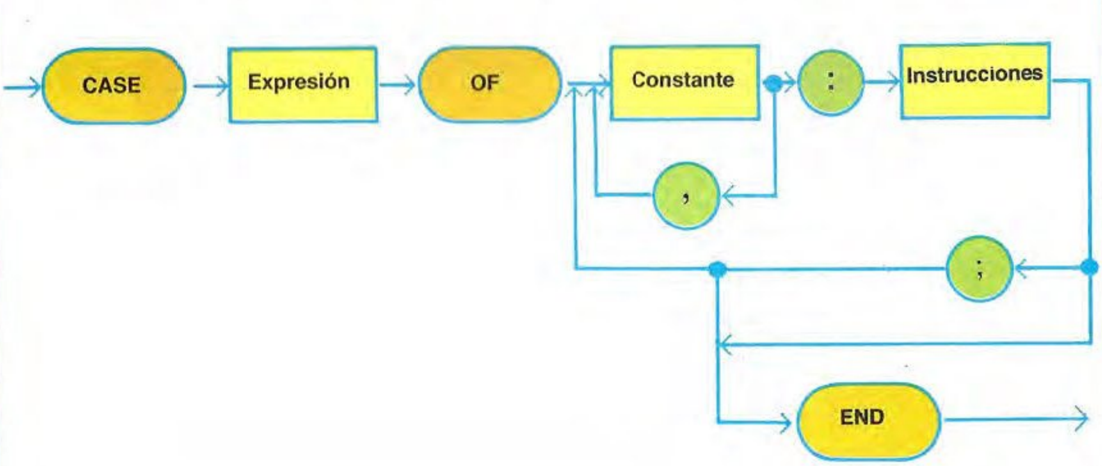
```
WRITELN ('INTRODUCIR FECHA DE NACIMIENTO');
READLN (DIA, MES, AÑO);
IF (DIA > 31) OR (MES > 12) THEN
  BEGIN
        WRITELN ('LA FECHA NO ES CORRECTA, REESCRIBELA');
        READLN (DIA, MES, AÑO);
  END;
ELSE
  BEGIN
        WRITELN ('LA FECHA ES CORRECTA Y ES:');
        WRITELN ('DIA=', DIA:5, 'MES=', MES:5, 'AÑO=', AÑO:6);
  END.
```

La instrucción CASE... OF... Esta instrucción puede considerarse una extensión de la IF... THEN... ELSE... Con ella es posible seleccionar entre varias instrucciones (simples o compuestas) una instrucción a ejecutar según el valor asumido por una determinada expesión. Sólo se ejecuta una sola de las posibles instrucciones alternativas; las otras se saltan.
El diagrama sintáctico se muestra abajo, mientras que en la página siguiente se muestra el esquema lógico. Una vez evaluada la expresión escrita inmediatamente después de la palabra reservada CASE, su valor se compara con el de las constantes (C1, C2, ..., CN) escritas después de la palabra reservada OF. Si el valor de la expresión es igual a la constante C1 se ejecuta la instrucción 1, si resulta igual a la constante C2 se ejecuta la instrucción 2, y así sucesivamente. También en este caso, las instrucciones a ejecutar pueden ser simples o compuestas. El segmento del programa indicado en la parte inferior de la página siguiente muestra el uso de la CASE... OF...
Si a la petición de introducción de un número se responde con el valor 1, se imprimirá la palabra UNO, si se introduce el 2 se imprimirá la palabra DOS, etc. La última línea muestra el caso en que a una misma instrucción se asocien más valores de control; entonces la instrucción se ejecutará si el valor de la variable es igual a una

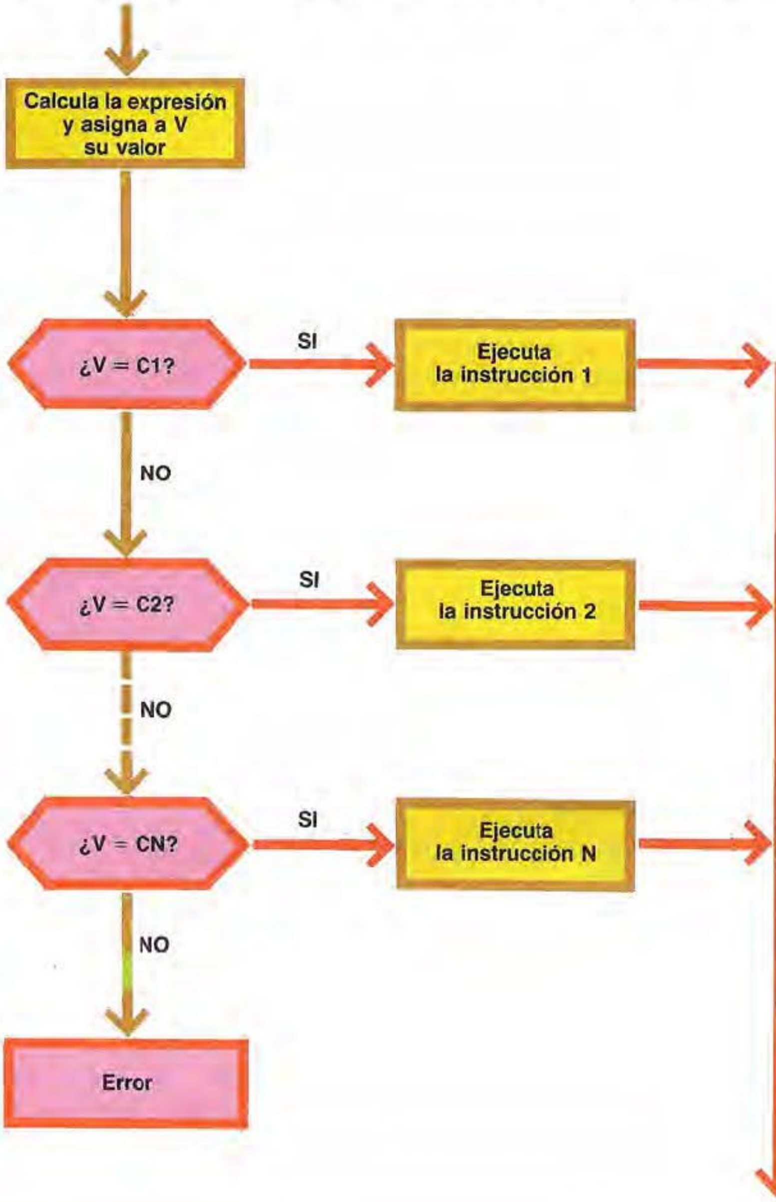


Circuitos integrados antes de su encapsulado en la matriz resinosa externa.

DIAGRAMA SINTACTICO DE LA INSTRUCCION CASE... OF...



ESQUEMA LOGICO DE LA INSTRUCCION CASE... OF...



APLICACION DE LA INSTRUCCION CASE... OF...

```
WRITELN ('INTRODUCIR UN NUMERO COMPRENDIDO ENTRE 1 y 4');
READLN (K);
CASE K OF
  1 : WRITELN ('UNO');
  2 : WRITELN ('DOS');
3,4 : WRITELN ('TRES O CUATRO');
END
```

cualquiera de las constantes contenidas en la lista asociada, mientras que si el valor de la variable de control no coincide con ninguno de los valores especificados (C1, C2, ..., CN), el resultado de la selección es indefinido y se genera un error. Siempre debe considerarse esta eventualidad en el uso de la instrucción CASE... OF... Efectivamente, el uso de la instrucción sólo es aconsejable cuando todos los valores que asumirá la variable de control se saben a priori. En este caso es posible insertarlas en la instrucción, y la selección nunca podrá ser indefinida. Cuando no es posible conocer anticipadamente todos los valores que la variable de control podrá asumir no conviene utilizar la instrucción CASE... OF...

La variable de selección no debe ser necesariamente entera, sino que puede pertenecer al tipo carácter o a otro tipo definido por el usuario. Por ejemplo, si se define el tipo de variable DIAS DE LA SEMANA, como se indica abajo, y si se considera la variable como perteneciente a tal tipo, es posible escribir la sección del programa indicada en la misma figura, que tiene como efecto la impresión de uno de los dos mensajes LUN y VIE o bien uno de los valores SAB o DOM. Obsérvese finalmente que la palabra clave END sirve para delimitar el final de la instrucción CASE... OF... y no para indicar el final de una instrucción compuesta o el final del programa.

La instrucción de salto incondicionado: GOTO...

La instrucción GOTO... permite alterar el flujo de ejecución de un programa de manera absoluta, es decir no condicionada al valor de ninguna variable de control. La instrucción de salto permite transferir el control de un punto a otro del programa, siempre que se haya identificado la instrucción que constituye el punto de llegada de dicho salto. Esto se obtiene con el uso de las etiquetas (labels). Las etiquetas, como en el Fortran, son números enteros, comprendidos normalmente en el intervalo de 1 a 9999, colocados delante de la instrucción a identificar y separados de la misma por medio del símbolo : (dos puntos). Por ejemplo, la instrucción

```
10:A: = A + 1
```

se identifica con la etiqueta 10.

Para ser válidos, los números utilizados en las etiquetas deben declararse al inicio del programa, en la declaración de LABEL. Esta declaración debe aparecer en la sección de las declaracione **antes** de las declaraciones de constantes y de variables. La sintaxis se muestra en la figura de la parte superior de la página siguiente. En la declaración deben relacionarse todos los números enteros que se utilizarán en el programa, por ejemplo así:

```
LABEL 10, 20, 30, 40;
```

Una vez definidas las etiquetas es posible describir la instrucción de salto incondicionado GOTO, cuya sintaxis se muestra en el segundo diagrama de la página siguiente. Por ejemplo, la instrucción

```
GOTO 100
```

transfiere el control, en el ámbito del programa, a la instrucción identificada por la etiqueta 100,

APLICACION DE LA INSTRUCCION CASE... OF...

Definición de tipo

```
DIAS DE LA SEMANA = (LUN, MAR, MIE, JUE, VIE, SAB, DOM);
```

Sección del programa

```
CASE HOY OF
LUN, MAR, MIE, JUE, VIE: WRITELN ('¡VE A TRABAJAR!);
SAB, DOM                : WRITELN ('¡POR FIN HA LLEGADO EL FIN DE SEMANA!');
END;(* FIN DE LA INSTRUCCION CASE... OF... *)
```

SINTAXIS DE LA DECLARACION DE LABEL



SINTAXIS DE LA INSTRUCCION GOTO...



que debe aparecer tanto en la declaración de etiquetas como en el cuerpo del programa.

La instrucción GOTO no se utiliza en Pascal como suele hacerse en otros lenguajes como el Basic y el Fortran. Esto es posible gracias a la notable flexibilidad de empleo de las instrucciones de control propias del Pascal, que permite hacer superfluo en la casi totalidad de los casos el uso de la instrucción GOTO. La programación estructurada, de la que el Pascal implanta los conceptos más válidos, tiende a eliminar el empleo de las instrucciones de salto incondicionado, cuyo uso indiscriminado se indica con el término «spagetti coding», a causa del elevado número de líneas entrelazadas que se produce. Y viceversa, un programa que utiliza sólo las estructuras de control de alto nivel puede definirse como un programa **estructurado**.

Aspectos particulares del Pascal

En este párrafo se examinarán por encima algunas particularidades del lenguaje Pascal, con el objeto principal de indicar a los lectores que estén interesados en profundizar en las características de dicho lenguaje.

El primer aspecto a subrayar corresponde a los **datos estructurados**.

En Pascal, como en los otros lenguajes, es posible definir las matrices. Los índices pueden ser de tipo lógico (BOOLEAN), carácter (CHAR), o constante.

Los siguientes son ejemplos válidos de declaración de matriz:

```
TYPE
  VECTOR = ARRAY (1.. 70) OF INTEGER;
  TABLA = ARRAY (BOOLEAN) OF CHAR;
  MATRIZ = ARRAY (1.. 3, 1.. 3) OF INTEGER.
```

Además de la operación de asignación, en los elementos de un array se definen todas las operaciones de comparación (menor, mayor, igual y sus combinaciones).

Los elementos de un array deben pertenecer necesariamente al mismo tipo. En el caso que se quiera crear una estructura de datos con elementos que no sean todos del mismo tipo, es posible utilizar el tipo **record**.

La definición del tipo record permite especificar para cada elemento, llamado campo (en inglés field), su tipo y un identificador. Por ejemplo, la ficha de un alumno puede estructurarse como en el listado de abajo; el identificador ALUMNO especifica una estructura de tipo record, mientras que NOMBRE es el identificador de un campo de tipo ARRAY y MATRICULA es el identificador de un campo de tipo INTEGER, etc.
Otro aspecto importante del Pascal corresponde al procedimiento. Se llama **procedimiento** o función de un bloque de instrucciones que puede reclamarse especificando su nombre, análogamente a lo que sucede para las subrutinas del Fortran. Un procedimiento se construye con las mismas reglas válidas para la construcción de un programa Pascal; es decir puede tener una parte declarativa, formada por LABEL, CONST, TYPE, VAR, PROCEDURE, FUNCTION, y una parte ejecutiva, encerrada entre las palabras reservadas BEGIN y END. El último aspecto a subrayar es la gestión de los files.
En Pascal, un **file** se define como un dato estructurado que contiene una secuencia de elementos, todos del mismo tipo, normalmente residente en un soporte magnético (disco o cinta).
En el Pascal estándar, la búsqueda en el interior de un file puede producirse sólo secuencialmente, a través del acceso a los records simples que componen el file.
Las siguientes son declaraciones de tipo file:

```
TYPE

   MATRICULAS = FILE OF INTEGER
   NOMBRES    = FILE OF CHAR
```

Sobre un file secuencial puede trabajarse en escritura (instrucciones REWRITE) o en lectura (instrucción RESET). En el primer caso es posible memorizar las informaciones en el file con la instrucción PUT, mientras que si el file se abre en lectura, puede accederse a los records con la instrucción GET.
La limitación de tener que acceder a las informaciones siempre, y sólo de modo secuencial, es muy pesada. Actualmente, muchas versiones del Pascal prevén también los files de acce-


Grazia Neri/Black Star

Placas de circuitos ensambladas en el interior de un monitor gráfico.

EJEMPLO DE DEFINICION DEL TIPO RECORD

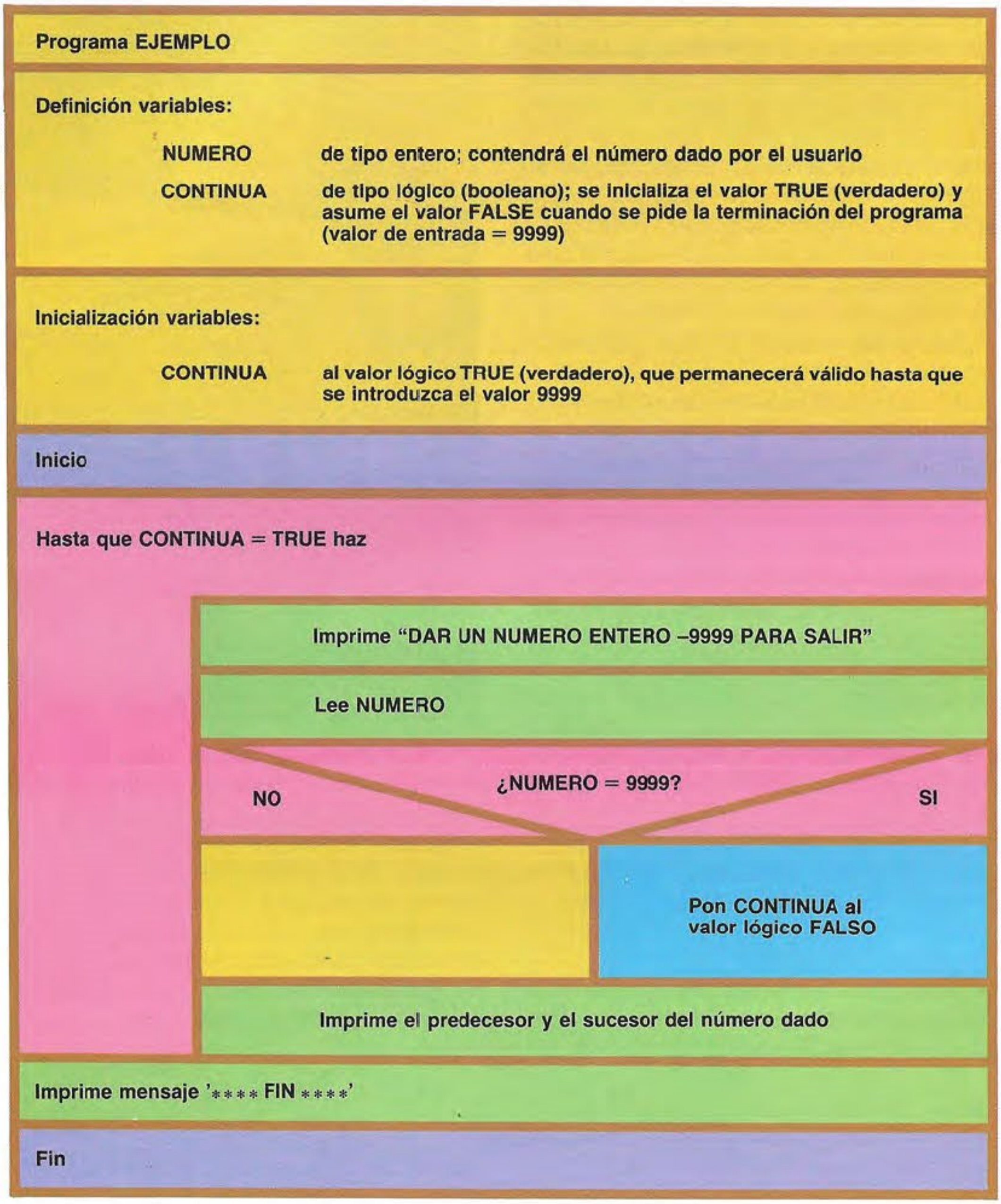
```
TYPE
  ALUMNO: RECORD
                 NOMBRE     : ARRAY (1 .. 10) OF CHAR;
                 APELLIDO   : ARRAY (1 .. 20) OF CHAR;
                 MATRICULA : INTEGER;
                 EDAD       : INTEGER;
                 DIRECCION : ARRAY (1 .. 30) OF CHAR;
                 TELEFONO  : INTEGER;
                 CLASE      : INTEGER;
          END;
```

so directo o de acceso indexado (indexed o keyed files), no previstos en la versión estándar.

Ejemplos de programación en Pascal

Aquí abajo se ha ilustrado el diagrama de flujo estructurado correspondiente a un ejemplo de aplicación sobre las funciones PRED y SUCC; la lista y salidas pueden verse en el listado de la página siguiente. En este caso se ha pensado pedir al usuario que introduzca un número entero en base al cual imprimir el predecesor y el sucesor del dato introducido.

USO DE LAS FUNCIONES PRED Y SUCC
DIAGRAMA DE FLUJO ESTRUCTURADO



USO DE LAS FUNCIONES PRED Y SUCC

```
*PROGRAM IMPRIME (INPUT,OUTPUT);

VAR
    NUMERO     : INTEGER;
    CONTINUA   : BOOLEAN;

 BEGIN
    CONTINUA  := TRUE;
    WHILE   CONTINUA DO
      BEGIN
        WRITELN ('DAR UN NUMERO ENTERO - 9999 PARA SALIR');
        READLN  (NUMERO);
        IF NUMERO = 9999 THEN CONTINUA : = FALSE;
        WRITELN ('NUMERO PRECEDENTE = ', PRED (NUMERO));
        WRITELN ('NUMERO SIGUIENTE = ', SUCC (NUMERO));
      END;
     WRITELN;
     WRITELN;
     WRITELN;
     WRITELN ('      ****  FIN  ****      ');
 END.

DAR UN NUMERO ENTERO - 9999 PARA SALIR
NUMERO PRECEDENTE =            3
NUMERO SIGUIENTE  =          5
DAR UN NUMERO ENTERO - 9999 PARA SALIR
NUMERO PRECEDENTE =           -11
NUMERO SIGUIENTE =           -9
DAR UN NUMERO ENTERO - 9999 PARA SALIR
NUMERO PRECEDENTE =           776
NUMERO SIGUIENTE =          778
DAR UN NUMERO ENTERO - 9999 PARA SALIR
NUMERO PRECEDENTE =           -2
NUMERO SIGUIENTE =           0
DAR UN NUMERO ENTERO - 9999 PARA SALIR
NUMERO PRECEDENTE =          9998
NUMERO SIGUIENTE =        10000


    ****  FIN  ****
```

El programa prosigue hasta que no se introduce el valor 9999, interpretado como terminador.

El control de fin de ejecución se efectúa de la instrucción WHILE... DO..., que permite repetir el ciclo hasta que el usuario decide salir.

Por tanto, las funciones PRED y SUCC se insertan directamente en la instrucción de impresión, y esto simplifica mucho el programa.

En la página siguiente se ilustra el diagrama de flujo estructurado de un programa que especifica el uso de las instrucciones READ y READLN. El correspondiente listado puede verse en la misma página, con una salida de demostración.

En el acto de la introducción de los primeros dos datos, apenas introducido el carácter alfanumérico, se imprimen los mismos datos; la segunda vez, para terminar la lectura y para tener la impresión, debe pulsarse la tecla RETURN.

Efectivamente, en el primer caso, los valores se leen hasta que no se ha agotado la lista de las variables de la READ, mientras que en el segundo caso, la lectura se cierra sólo por la introducción del carácter CR (Carriage Return, retorno del carro).

Si utilizando la READLN se introducen más valores que los necesarios, el programa examina sólo los que agotan la lista de los parámetros (en el ejemplo sólo el primer número y el primer carácter), mientras que los otros se ignoran.

En el diagrama de flujo estructurado y en el listado de la página 1265 finalmente se ilustra un ejemplo de uso de la instrucción WHILE... DO...

APLICACIONES DE LAS INSTRUCCIONES READ Y READLN

Programa EJEMPLO

Definición variables:

NUMERO	**de tipo real**	**valores dados por el usuario**
CARACTER	**de tipo carácter**	

Inicio

- **Imprime mensaje de petición de un número y de un carácter**
- **Lee un número y un carácter**
- **Imprime los valores leídos**
- **Imprime mensaje de búsqueda de un número y de un carácter**
- **Lee en el interior de una línea <u>un</u> número y <u>un</u> carácter**
- **Imprime el número y el carácter leídos**
- **Imprime mensaje **** FIN ******

Fin

APLICACIONES DE LAS INSTRUCCIONES READ Y READLN

```
*PROGRAM ESCRIBE (INPUT,OUTPUT);

VAR
     NUMERO         : REAL;
     CARACTER       : CHAR;

BEGIN
    WRITELN ('DAME UN NUMERO Y UN CARACTER' );
    READ     (NUMERO,CARACTER);
    WRITELN;
    WRITELN  ( NUMERO:5:2,'    ',CARACTER:1);
    WRITELN  ;
    WRITELN  ;
    WRITELN ('DAME UN NUMFRO Y UN CARACTER' );
    READLN  ('NUMERO, CARACTER);
    WRITELN ;
    WRITELN (NUMERO:5:2,'     ',CARACTER:1);
    WRITELN ;
    WRITELN ('     ****   FIN   ****     ');
END.

DAME UN NUMERO Y UN CARACTER

45.00    t


DAME UN NUMERO Y UN CARACTER

67,80    H

     ****   FIN   ****
```

EJEMPLO DE USO DE LA INSTRUCCION WHILE... DO...

Programa EJEMPLO

Definición constantes:

PUNTO	**equivalente al carácter '.'**

Definición variables:

INDICE	**contador de los caracteres leídos; de tipo entero**
CARACTER	**contiene los caracteres dados por el usuario; de tipo carácter**

Inicialización variables:

INDICE	**se pone inicialmente a 0**

Inicio

Lee CARACTER

Mientras CARACTER es diferente de PUNTO haz

- **Incrementa en 1 el contador de caracteres INDICE**
- **Lee un CARACTER**

Imprime el número de caracteres leídos (excluido el punto)

Imprime mensaje de fin de programa

Fin

APLICACIONES DE LA INSTRUCCION WHILE... DO...

```
PROGRAM NUMCARAC    (INPUT,OUTPUT);

CONST
      PUNTO = '.';

VAR
      INDICE   : INTEGER;
      CARACTER : CHAR;

BEGIN
    INDICE      :=0;
    WRITELN     (' ESCRIBE UNA FRASE - TERMINADA CON UN PUNTO');
    WRITELN;
    READ        (CARACTER);
    WHILE CARACTER < > PUNTO DO
      BEGIN
        INDICE    :=  INDICE + 1;
        READ (CARACTER);
      END;
    WRITLELN;
    WRITLEN ( ' EL NUMERO DE CARACTERES EN LA LINEA ES' ' ',INDICE);
    WRITELN;
    WRITELN ( '      ****   FIN   ****      ' );.
END.
```

```
ESCRIBE UNA FRASE - TERMINADA CON UN PUNTO


EL NUMERO DE CARACTERES EN LA LINEA ES          36

   ****  FIN   ****
```

Recapitulación: desarrollo de un proceso de facturación

Este ejemplo de aplicación se ha elegido porque permite analizar las principales funciones que normalmente se piden a un procesador. En particular, examinaremos un complejo procedimiento de gestión de los files y de las máscaras, junto con las técnicas de redondeado sobre valores numéricos. Se trata de un ejemplo al que pueden reducirse numerosos casos, que no necesita ningún conocimiento específico y es una recopilación de todo lo expuesto hasta ahora.

Análisis del problema

El objeto del proceso es gestionar una serie de transacciones que determinan los movimientos de artículos del almacén y de los respectivos importes de venta o de compra. Sobre estos dos términos (ventas/compras) se articulan todos los flujos de los datos.

Una actividad cualquiera de naturaleza comercial necesita como mínimo la gestión de los files

— Clientes
— Proveedores
— Almacén
— Caja

En los files Clientes y Proveedores se memorizan los datos y los importes (haber para los clientes, debe para los proveedores); el file Almacén contiene las descripciones de las mercaderías objeto del movimiento y el file Caja se utiliza para mantener actualizada la situación económina.

Exposición de sistemas de proceso Apple.



G. Picchetti/Marka

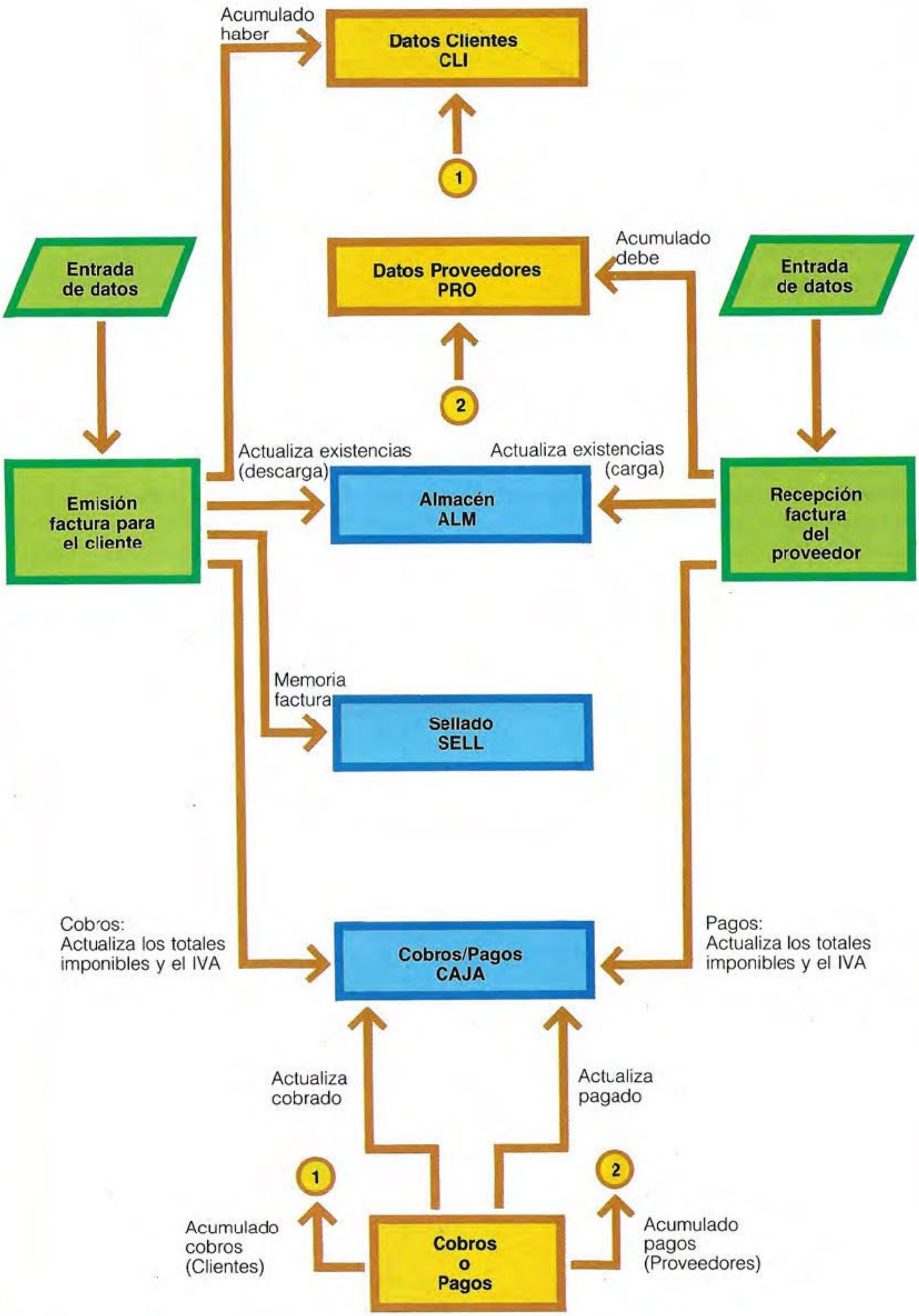
El contenido y la propia presencia de estos files deben considerarse de manera elástica; por ejemplo, el file Almacén podría no estar presente o contener una sola denominación ficticia para todas aquellas actividades que no tienen denominación real de mercadería (servicios, asesoramientos, etc.).
El documento que origina un movimiento de mercaderías o de dinero es la factura. En ella deben indicarse los siguientes datos:

— datos del cliente
— mercaderías e importes correspondientes
— IVA e importes correspondientes

Los primeros dos campos tienen un significado obvio, mientras que el último necesita algunos comentarios.
El IVA* es un impuesto que se aplica en el ámbito de la CEE. Se calcula como una parte porcentual sobre el importe de una transacción; el valor depende del género de artículo objeto de la misma. Puede suceder que en una misma factura deba calcularse el importe total del IVA como suma de diversos importes, cada uno obtenido con la aplicación de un porcentaje diferente (en relación al tipo de mercadería) y a un determinado importe.
La factura puede **emitirse**, y en tal caso corresponde a una venta (de un bien o de un servicio), o puede **recibirse**, y en tal caso se trata de una compra o de un servicio recibido.
En base a los datos contenidos en las facturas emitidas debe disminuirse la existencia del almacén (las mercaderías salen), y al mismo tiempo debe aumentarse el importe del cliente que se refiere a la voz de los cobros. Y viceversa, los datos contenidos en las facturas recibidas deben incrementar la existencia del almacén (es una mercadería que entra), incrementar el importe debido al proveedor e incrementar la voz de los gastos.
Además de estos movimientos de carácter de gestión, deben memorizarse los datos correspondientes a las cargas fiscales, en particular a los importes del IVA.
En el argot comercial, el IVA se llama «una partida de giro», o sea un importe que puede ser transferido de una a otra de las entidades que intervienen en las sucesivas transacciones. Por ejemplo, al recibir una factura de un proveedor con un importe 1000 y 20 de IVA (2%) el debe será 1020, del cual 1000 (imponible) es el pago del bien y 20 es el impuesto que el proveedor deberá pagar a su vez.
Si este bien se revende, supongamos a 1500, el correspondiente importe IVA será 30 (mismo porcentaje) y el comprador deberá pagar 1530. De este importe IVA (30), en realidad una parte ya se ha pagado (20 entregado al proveedor) y , por tanto, el debe IVA es 10 (30 – 20).
De todo lo dicho es necesario que el procedimiento ponga en evidencia los importes IVA cobrados al cliente y los pagados al proveedor, para poder calcular la diferencia. El flujo general del procedimiento se ha esquematizado en la página siguiente.
Con respecto a lo expuesto aparece la nueva función Cobros/Pagos, que en realidad no está ligada a la facturación, pero de la que constituye un útil complemento.
La emisión de una factura genera un movimiento de importes sólo desde el punto de vista contable. En lo que respecta a los gastos fiscales, la cifra indicada en una factura emitida es como si se hubiese cobrado, mientras que en realidad, el pago puede producirse más tarde o también puede ser fraccionado. Por estos motivos es útil gestionar la situación Cobros/Pagos, que se refiere a un movimiento real de caja y muestra la liquidez, el importe de los créditos y el importe de las deudas.
La gestión de la caja no comporta obligaciones ni formalidades particulares, puesto que puede estructurarse según las necesidades o costumbres específicas sin ningún vínculo. El único objetivo es mantener una visión actualizada del flujo de dinero de entrada y de salida.
El conocimiento de una situación global en el debe y en el haber puede no ser suficiente: deben identificarse los clientes o los proveedores respectivamente en el debe y en el haber. Por tanto, los cobros y los pagos, además de producir movimientos de caja, deben actualizar los datos correspondientes a cada cliente o proveedor unitario.
Finalmente debe considerarse la eventualidad de una devolución de mercadería. En este caso existen dos posibilidades. Si la factura correspondiente ya se ha emitido, deben devolverse los correspondientes importes con una «nota de abono»; si la factura no se ha emitido, sólo deben reintegrarse las mercaderías al almacén.

* Importe sobre el Valor Añadido, de futura aplicación en España.

ESQUEMA DEL PROCEDIMIENTO DE FACTURACION



Cómo se proyecta un videojuego inteligente (2)

Implantar un programa (que hasta ahora hemos llamado con un poco de presunción autómata) significa construir una fuente de señales de control que gobierne centralizadamente las manifestaciones del autómata hacia el exterior: las fichas. Para no continuar definiéndolo como autómata o adversario artificial, daremos un nombre al programa que vamos a construir.

Es un personaje de los dibujos animados norteamericanos que se llama Fred. Es un león simpático y bonachón que acepta con resignada paciencia todos los experimentos que hace sobre él un científico amigo suyo algo despistado.

Bien, llamaremos Fred a nuestro adversario artificial, con la esperanza de que también tenga aquellas dotes de comprensión y paciencia ante las vejaciones de que le haremos objeto.

Ahora sabemos que Fred estará construido con el fin de responder válidamente a una serie de jugadas nuestras efectuadas según las reglas simplificadas del Go que ya hemos visto.

Pero Fred no sólo está compuesto de reglas del Go; también las interpreta dinámicamente, utilizando a tal fin los instrumentos activos de su naturaleza de programa por procesador.

Con el fin de facilitar su interpretación en un lenguaje cualquiera, describiremos con detalle una serie de procedimientos expresados en razón de los criterios de abstracción funcional.

El uso de esta metodología permite conseguir diferentes objetivos primarios.

Presentando en primer lugar las relaciones de precedencia causal y no las de precedencia temporal, la metodología elegida facilita la comprensión por parte del lector de los mecanismos lógicos que regulan el juego del Go y que sirven para la construcción de Fred.

Como no se propone una mera copia del listado, sino su producción por parte del lector, este último podrá efectuar el proceso de emisión de la información. *Este fenómeno es una de las bases más importantes y fundamentales en la disciplina del aprendizaje, y tiene un gran valor pedagógico.*

Los procedimiento no se expresarán en un lenguaje de programación, sino en un castellano estructurado que corresponda a criterios de universalidad y permitirá una implantación en diferentes lenguajes, a elección del usuario.

Aquí se ha hecho virtud de la necesidad, porque también el Basic está en continua transformación y mejora: las nuevas versiones recientemente aparecidas en el mercado del software inducen a pensar que se tienen un progresivo acercamiento hacia el Pascal, a través de una implantación «procedurizada» del Basic.

Finalmente, hay la posibilidad de utilizar un lenguaje que se presta muy bien a la construcción de Fred: el Logo. Este lenguaje se comercializa en diversas versiones adaptadas a los diversos ordenadores personales y puede adquirirse fácilmente en las tiendas especializadas.

El autor ha utilizado la versión A1.5 del Logo sobre el procesador MPF3 de Digitek para implantar una serie experimental de procedimientos para el estudio y el desarrollo de Fred.

Construir Fred en realidad también significa aprender a jugar bien al Go, aplicando correctamente sus reglas y dominando la dinámica de las jugadas en el Go Ban.

Así queda claro que la casi totalidad de los procedimientos que vamos a definir tendrá el objetivo de describir cada vez una parte del ejercicio del juego, que alternativamente podrá ser gestionada tanto por nosotros mismos como por el propio Fred.

A medida que se vaya completando su implantación, si bien actuando en base a mecanismos lógicos muy rígidos, si está implantado correctamente propondrá jugadas no estúpidas, e incluso dotadas de aquel mínimo de imprevisión que harán de él un adversario simpático.

Entrando en el caso concreto, la primera operación que debemos realizar es la construcción del Go Ban y de las fichas, los objetos a través de los cuales tendremos la representación de las jugadas efectuadas.

Este programa sólo corresponde a la gestión de la pantalla; ésta no forma parte de Fred, pero es su complemento. Por tanto, lo describiremos en primer lugar, con el fin de aclarar cuáles son las exigencias de presentación del juego, a las cuales también deberá atenerse Fred.

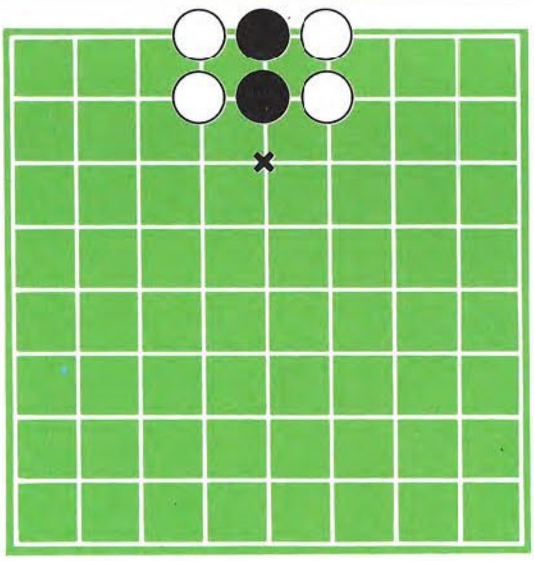
Recordaremos que el Go Ban, que en realidad es una rejilla cuadrada de 19 × 19 intersecciones por lado, y en nuestro caso, por motivos de simplificación del juego se reducirá a dimensiones más pequeñas: 9 × 9 intersecciones.

El procesador la presentará en la pantalla y deberá proporcionar algunas informaciones esenciales: la posición de las piezas en el Go Ban (así como el número de las fichas todavía dispo-

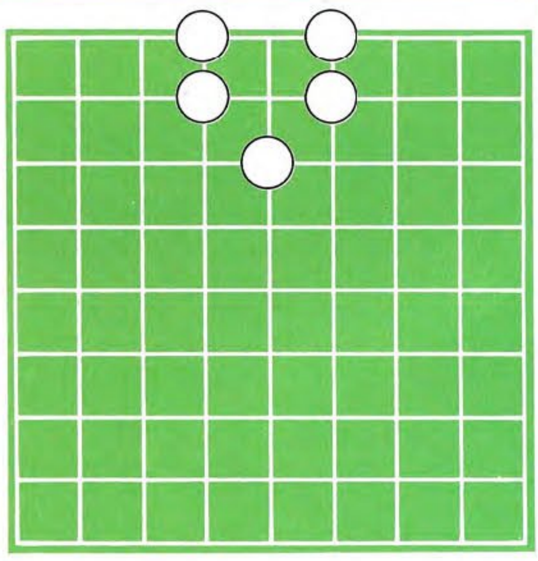
nibles por cada jugador), la indicación correspondiente a las fichas hechas prisioneras por cada contendiente y la eventual declaración de ATARI por parte de uno de los jugadores.
Efectivamente, llega un momento de la partida en que un grupo de conexión posee una sola libertad y uno de los jugadores se apresta a capturarlo. Como en el ajedrez (declaración de jaque), también en el Go, el jugador debe advertir al adversario de sus propias intenciones ofensivas, a las cuales el oponente, si puede, debe rehuir con la oportuna jugada.
La figura de abajo es un ejemplo en la que es necesaria la declaración de atari.
Y después, como se hace para el ajedrez, una identificación de las filas y de las columnas para poder direccionar la colocación de las fichas.
El problema de la representación del Go Ban es el primer escollo que debemos superar y que no tiene una resolución inmediata.
La construcción física de la rejilla puede realizarse en base a diferentes filosofías que, marcarán los caracteres el programa e influirán en los criterios de construcción de Fred.
Para ello desarrollaremos algunas consideraciones útiles a los fines de una mayor implantación del trabajo de programación: no hay un criterio particular de prevalencia de una filosofía sobre otra, sino el referido a una mayor o menor facilidad de extensión del listado.

El Go no es un videojuego con continuo movimiento, ni una representación estática e inmóvil: por el contrario, está dotado de un mínimo de acción, y precisamente en la descripción de ésta se evidencia la historia de la partida.
El programa ideal debería facilitar la reevocación de todas o gran parte de las jugadas según dos posibilidades. La primera es la de memorizar la representación gráfica de una jugada (es decir, la situación del Go Ban en un determinado momento) y construir, en el programa que gestiona el juego, un área en la que depositar, marcándolos en orden temporal, todos los datos videográficos que aparecen en la pantalla.
Reclamándolos ordenadamente en orden temporal creciente o decreciente, se tendrá una descripción total o parcial de la partida.
Con un análisis más profundo, esta solución parece ser demasiado burda: efectivamente, ocupa inúltilmente mucha área de memoria, no lleva a ningún algoritmo interactivo y no estimula la fantasía del programador. En una palabra, no es optimizada.
De ahí la segunda propuesta en la que encontraremos dos elementos constitutivos integrados en ella. El primero es una rutina que considera la colocación en el Go Ban de una ficha tal como sugiere una jugada del jugador o de Fred, marcando la codificación de manera que resulte temporalmente identificada en nuestra historia

La posición alcanzada en el Go Ban muestra dos fichas negras en atari.



El blanco ha jugado capturando las fichas negras.



de la partida, después de la cual direcciona su memorización en una determinada área disponible en el programa, evalúa la situación en el Go Ban y reclama una segunda rutina. En ella encontraremos implantado un algoritmo que dibuja el Go Ban y las piezas en la pantalla.
Dadas las funciones de ésta, el programa no está obligado a almacenar las fichas, sino que es el mero representador gráfico.
Todavía subsiste el problema de la entrada de la ficha preseleccionada. Ésta puede suministrarse al procesador a través de una descripción alfanumérica en la que se han expresado las referencias cartesianas de un cruce del Go Ban (asignando un código de identificación a cada fila y columna), o bien a través de la manipulación de varios dispositivos, como el ratón, la mesa gráfica, el joistick. Ahora ha llegado el momento de realizar la construcción de Fred. Como se ha indicado en las evaluaciones previas hechas sobre las rutinas de representación, Fred no será un programa rígido e interminable; en cambio tomará vida de algunos procedimientos que, debidamente indicados y definidos, se reclamarán de un sistema de control variable, de manera que permita a Fred una discreta agilidad en la formulación de las jugadas.
La primera rutina permitirá a Fred leer la situación de las fichas sobre el Go Ban para construirse un mapa del juego.
Fred debe interpretar correctamente el significado de las figuras del juego para realizar a continuación una actividad de evaluación y de propuesta de su jugada. A tal fin, la rutina MAPA le permitirá identificar los grupos de conexión y las fichas que los constituyen (recordemos que éstas van de la simple ficha hasta las aglomeraciones de grandes dimensiones), buscando e indicando para cada una de ellas las correspondientes libertades. Para cada punto del Go Ban, Fred desarrollará el procedimiento de identificación sobre:

- una ficha cada vez en el grupo de conexión
- una libertad cada vez del grupo de conexión

y sistemáticamente la búsqueda de una ficha del otro color en las adyacentes del grupo de conexión considerado.
Fred desarrollará la búsqueda a través de un camino en el interior del grupo; haciéndolo así fijará en su memoria una representación correcta de las fichas en el Go Ban.
Esto sucede situando significativamente los flags correspondientes en aquel punto del Go Ban y referidos a los diversos parámetros de identidad de la ficha examinada con los atributos típidos (libertad, inserción en un grupo, etc.) que ésta presenta en aquel momento. Es importante observar que el punto de omisión de Fred son los cruces y no las piezas, de otra manera MAPA no analizaría sistemáticamente todos los cruces del Go Ban utilizando el algoritmo de MAPA, después de lo cual será el propio MAPA el que informará a Fred acerca de la situación existente en el tablero.
MAPA es un grupo de reglas de comportamiento (práctica de búqueda) que guían a Fred en la primera parte de la formulación de su respuesta ante una jugada nuestra.
Con la imaginación nos meteremos en el interior de MAPA y analizaremos en detalle su comportamiento. Cuando MAPA ha entrado en el Go Ban y en su interior ha identificado una ficha, busca, a través de un mecanismo de explora-

DESCRIPCION DEL MODULO MAPA

```
MAPA (color, x)
INICIO
   SI hay una ficha en x E
      es del color «color» E
      no se ha marcado
   ENTONCES
      INICIO
         Márcala
         RECLAMA MAPA (color, NORTE (x))
         RECLAMA MAPA (color, ESTE (x))
         RECLAMA MAPA (color, SUR (x))
         RECLAMA MAPA (color, OESTE (x))
      FIN
   DE OTRA MANERA
      INICIO
         Si no hay ficha en x
         ENTONCES
           INICIO
              Marca el punto como libertad
              Incrementa el contador de las
              libertades
           FIN
      FIN
FIN
```

Con el módulo MAPA Fred busca, encuentra y cuenta las libertades de un grupo de conexión que contiene una ficha del color «color» en el punto x. MAPA se reclama a sí mismo repetidamente, memorizando x.

ción circular, los cruces libres alrededor de ella y las marca con sus libertades, incrementado también el valor global en el registro de las libertades de aquel grupo. Siempre con el fin de dar a Fred el mayor número de datos significativos el módulo MAPA actúa utilizando un criterio recursivo: se reclama a sí mismo tantas veces como es útil, lo que le permite avanzar paso a paso en el interior del grupo de conexión, determinando así su estructura y sus confines.

Es fundamental que MAPA consiga reconocer los lados exteriores de Go Ban por dos motivos:

a) para que Fred no deposite fichas fuera del Go Ban a continuación de una indicación errónea
b) para que Fred pueda gestionar bien las «fichas fantasma», que revisten una importancia estraetégica fundalmental en las jugadas de ángulo.

Ahora que Fred posee los instrumentos de búsqueda para interpretar las posiciones de las fichas sobre el Go Ban, puede ponerse a formular una respuesta a nuestra jugada.

Entre los dos jugadores, Fred es considerado el más débil: por esto se le adjudica el beneficio de un handicap, consistente en algunas fichas con las cuales debe proponer una correcta apertura de partida. Siempre por la misma (presunta) debilidad, y en homenaje a la antigua tradición del Go, podrá tener la gracia de mover en primer lugar y con las fichas negras.

El procedimiento a través del cual Fred realiza estas tareas de juego tomará el nombre de JUEGA. Éste estará compuesto de algunas rutinas que realizarán una serie de tareas funcionales precisas dedicadas al análisis de determinados ámbitos de juego.

La primera acción sugerida por el módulo JUEGA es la colocación de las fichas de handicap. Se trata de un instrumento de compensación adecuado para reequilibrar artificialmente las fuerzas en el campo: si la diferencia entre las habilidades de los dos jugadores fuese muy marcada, ya no existiría el placer de jugar una partida, puesto que el éxito quedaría fuertemente condicionado a favor del más fuerte. Pero el Go es el juego de la inteligencia y del sutil equilibrio entre las partes. En su naturaleza de juego de posición, que principalmente intenta establecer una posición de dominio seguro sobre una parte del territorio, asume un papel de privilegio que será difícil compensar por parte del adversario. Concediendo esta facultad al jugador más débil, se reequilibran las posibilidades de la partida, haciendo estable y profundo el interés de ambos contendientes y poder jugarla con astucia e interés.

Nuestra capacidad de jugar al Go probablemente no es tan superior a la de Fred. No obstante, se establece un handicap a su favor, por lo menos en razón de la presunta superioridad del hombre sobre la máquina.

Como el Go Ban utilizado es de 9 × 9 intersecciones, aquí no es posible aplicar la regla japonesa de las «estrellas» (hoshi), que establece un riguroso orden de posicionado de las fichas de handicap en otras tantas posiciones. Ahora haremos referencia a la tradición del juego chino, que establece en este caso la libre disposición de las fichas de handicap en cualquier localización del Go Ban.

El número de las fichas de handicap varía de 1 a 9 según la relación de fuerzas existente entre los jugadores: en nuestro caso seremos propensos a adjudicar a Fred 5 o 7 fichas. Como esto es más del 10% de las fichas a su disposición, Fred podrá adquirir así una posición estratégica válida, especialmente si elige efectuar una apertura de ángulo.

(continúa en la pág. 1295)

DESCRIPCION DEL MODULO JUEGA

```
JUEGA
INICIO
   Dispón las fichas negras de handicap
   MIENTRAS dura la partida HAZ
      INICIO
         Presenta el Go Ban
         Adquiere ficha Blanco
         (introducida por teclado)
         RECLAMA EFECTO FICHA BLANCO
         (por efecto fecha del Blanco)
         RECLAMA EFECTO FICHA NEGRO
         (por intento ficha del Negro)
         RECLAMA MODELOS
         (para buscar un modelo de encuentro)
         Coloca la ficha negra
      FIN
FIN
```

El módulo JUEGA es el cuerpo central del programa que hemos llamado Fred. A él se refieren muchas subrutinas que ejecutan, en su interior, tareas específicas.

Files utilizados

En la figura de la página siguiente se han representado los records de los principales files utilizados. Para todos ellos se ha previsto un campo inicial de tres caracteres (bytes 1 a 3) que contiene el código de identificación. En la versión Basic, este código no sería necesario suponiendo una estructura random con código igual al número del record: el primer artículo de almacén introducido ocupa el record 1 y tiene código 1, mientras que el segundo ocupa el record 2 y tiene código 2, etc. Este campo también se ha introducido pensando principalmente en la utilización del procedimiento en Cobol, pero asimismo para dar una mayor elasticidad al programa: el usuario final ya podría tener una codificación propia, y ser remiso a asumir una nueva. Por ejemplo, si en la gestión de un almacén no se utiliza el código exterior, cada artículo se codifica con el número acumulativo de introducción (número de record), mientras que puede resultar más cómoda una sigla, no necesariamente numérica, que indique en forma sintética de qué tipo de mercadería se trata.

Sin embargo, en este segundo caso aparecen complicaciones. Utilizando como código el número de record, la lectura de los datos inherentes a un artículo es inmediata, puesto que la posición del record en el interior del file está indicada por el código. Y viceversa, codificando los artículos con siglas arbitrarias, cada vez es necesario realizar una búsqueda. En el caso de pocos artículos, el tiempo necesario podría ser aceptable, pero más allá de un determinado número de artículos, es indispensable que el almacén esté ordenado por códigos, de manera que se pueda realizar una búsqueda rápida, por ejemplo de tipo binario.

El número de artículos para el cual conviene utilizar uno de los dos métodos alternativos (búsqueda secuencial, búsqueda en file ordenado) difícilmente puede evaluarse a priori; es necesario analizar cada caso, sobre todo a la luz del lenguaje particular y del hardware utilizados.

Para mantener una congruencia de formato, el campo del código también se ha previsto en los files en que no es estrictamente necesario, donde debe considerarse como un filler.

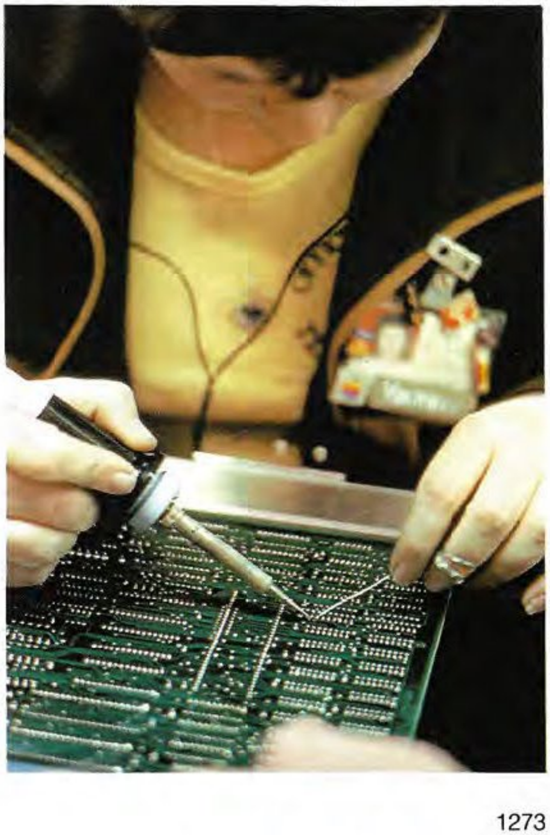
Los esquemas de record representados en la página siguiente se refieren sólo al área de datos. Los eventuales records de directorio deben gestionarse separadamente, con campos de diferentes significados (por ejemplo, número de los records ocupados, longitud total, etc.), con la advertencia de que en Basic normalmente los records deben ser de longitud fija; por tanto, también los del directorio deberán tener la misma longitud del record de datos. Los números indicados en la figura sobre cada uno de los records representan la longitud en bytes y la posición del campo. Por ejemplo, el campo descripción presente en el record del file ALM es de 20 caracteres de longitud y ocupa en el record los bytes de 4 a 23. Esta forma de representación gráfica es muy útil tanto en fase de detallado como en fase de depuración.

En la tabla de la pág. 1275 se han representado las descripciones de cada file con respecto al contenido y a la utilización.

Menú principal

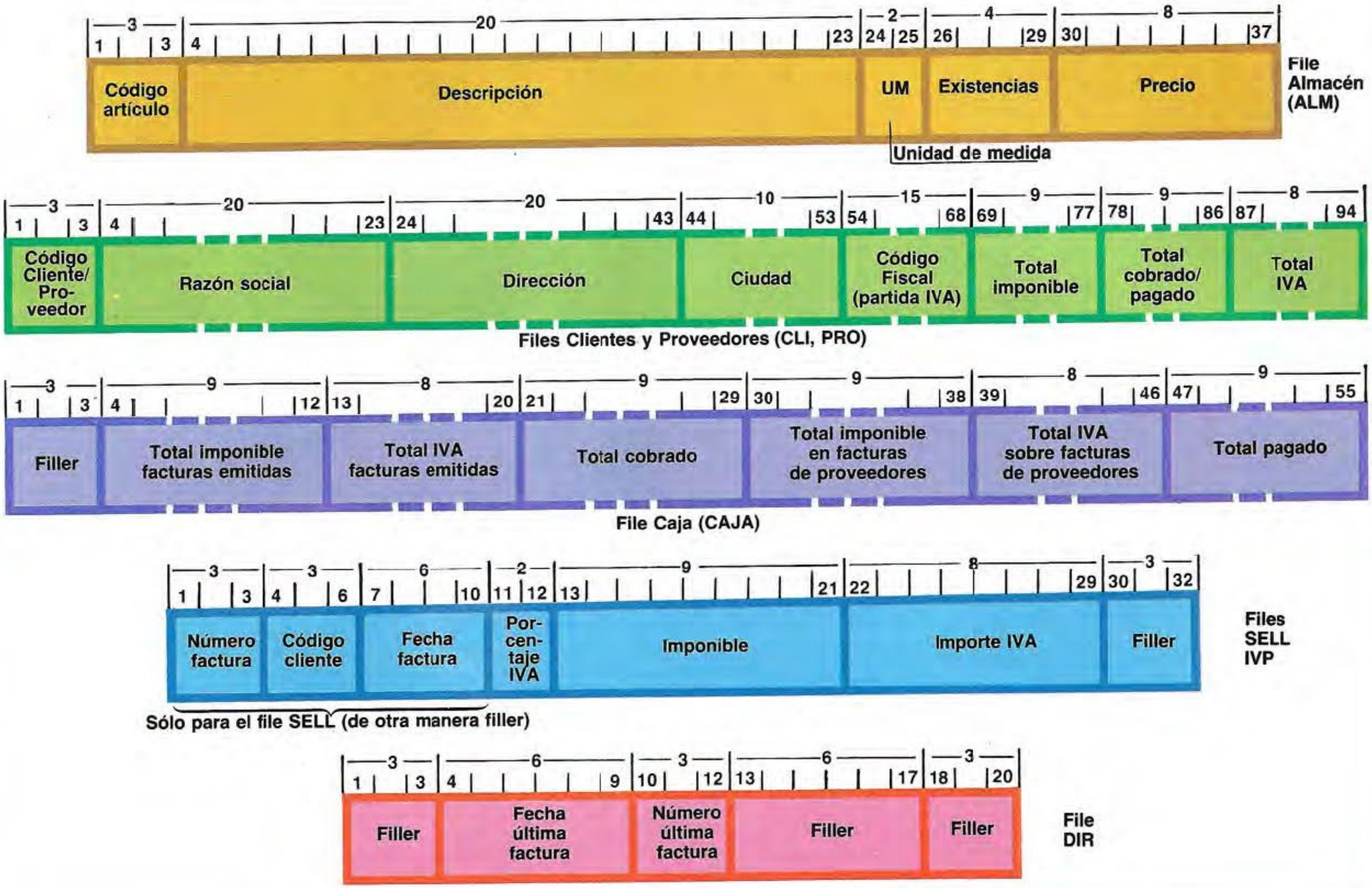
En la figura de la pág. 1276 se ha representado el diagrama de flujo del menú principal del procedimiento. No existen particularidades notables, por lo que el correspondiente listado se ha omitido. En la pág. 1277 se muestra el esquema

Montaje de la tarjeta madre de un ordenador personal.



J.P. Laffont/Grazia Neri-Sygma

ESQUEMA DEL RECORD DE LOS FILES



FILES UTILIZADOS EN EL PROCEDIMIENTO DE FACTURACION

Nombre	Longitud record en bytes	Descripción
DIR	20	Contiene informaciones de carácter general, como la fecha y el número de la última factura emitida. La numeración de las facturas debe ser progresiva; por tanto, debe memorizarse el último número utilizado para emplear el siguiente. La fecha puede ser útil como control
ALM	37	Contiene los datos de almacén y los correspondientes a las existencias y a los precios de los artículos. El campo UM (unidad de medida) puede ser útil sólo en los casos en que se hayan previsto diversos tipos de mercaderías. El campo Existencias es decrementado con las facturas emitidas e incrementado con las recibidas.
CLI, PRO	94	Los dos files son idénticos. Contienen los datos (de los clientes uno y de los proveedores otro), el total imponible (suma de los importes de cada factura), el total IVA y el total cobrado o pagado (actualización con el procedimiento Cobros/Pagos)
CAJA	55	Contiene los totales correspondientes al imponible y al IVA, a las facturas y a los movimientos cobrado/pagado. Estos mismos datos pueden obtenerse como suma de los importes contenidos en cada record sencillo de los files CLI y PRO. El total en un file separado permite la visión inmediata de la situación económica. El file contiene un solo record de datos
SELL	32	Contiene un resumen del importe IVA. Su presencia está ligada a la necesidad de imprimir un documento fiscal (registro IVA) en el que deben aparecer estos datos. Después de la impresión, los datos se anulan y el file puede reutilizarse. Normalmente, la impresión se produce cada 3 meses, por lo que la longitud máxima del file en records debe ser igual al número de facturas previsto en el ámbito de 3 meses
IVP	32	Tiene un formato análogo a SELL, pero contiene los importes acumulados y el IVA divididos por partes alícuotas. Se utiliza para conocer la situación conjunta a final de año

lógico completo. Del menú principal, introduciendo el número correspondiente a la función deseada, se tiene la carga del correspondiente menú secundario. Una ulterior selección entre las voces previstas en el menú secundario carga y envía a ejecución la parte deseada del procedimiento. Esta estructura permite una fácil segmentación del programa; cada procedimiento se configura como un bloque estanco que no intercambia datos con los demás. Esta lógica es muy usada en Basic, mientras que para otros lenguajes, o en máquinas más potentes que una personal, es el Sistema Operativo que gestiona la segmentación de los programas de manera totalmente transparente para el usuario. Sin embargo, también en este caso es de mucha utilidad subdividir todo el programa en un cierto número de procedimientos orientados a la solución de uno entre los diversos aspectos del problema.

Las ventajas que se derivan son dos:

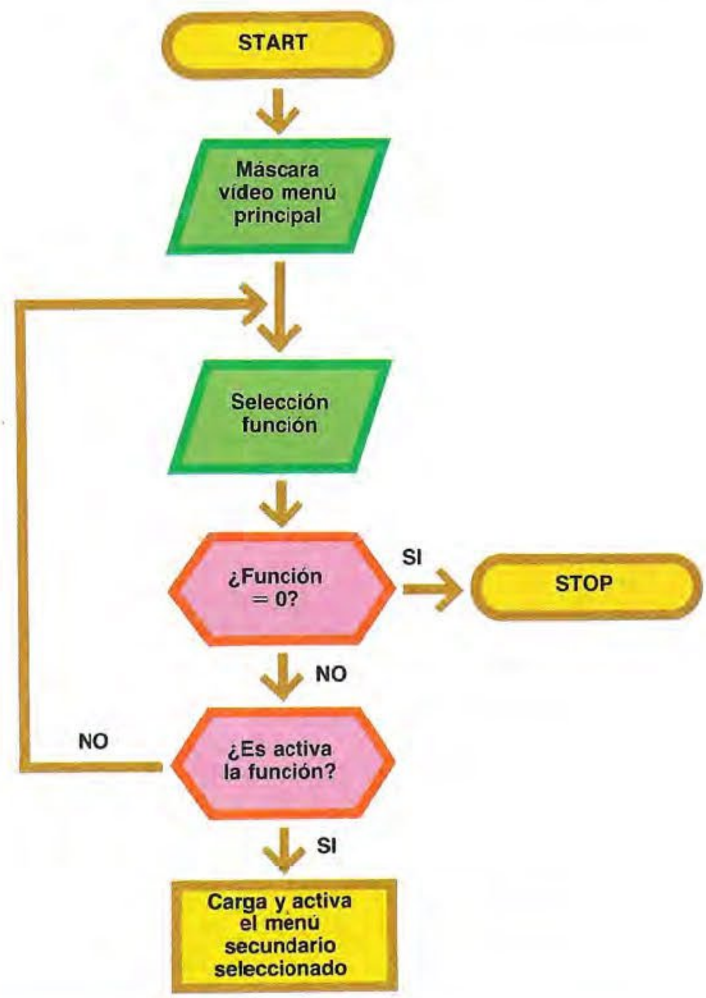
- Modularidad. El programa puede adaptarse a situaciones y a exigencias diversas eliminando o añadiendo procedimientos. Por ejemplo, la gestión del almacén (o una de

sus partes) puede eliminarse anulando el correspondiente procedimiento.

- Facilidad de mantenimiento y de depuración. Cuando se localiza un error puede aislarse y corregirse fácilmente. Las nuevas implantaciones se obtienen sustituyendo o modificando el correspondiente procedimiento, sin interaccionar con los otros.

En los sistemas menos evolucionados, esta técnica obliga a duplicar todas las rutinas utilizadas por más procedimientos (a menos que se adopten complicados sistemas de gestión que están totalmente a cargo del usuario), mientras que con los Sistemas Operativos más avanzados es posible utilizar una sola copia del módulo común, dado que su carga y las conexiones con las otras partes del procedimiento están gestionadas por el sistema y no necesitan intervenciones. En términos prácticos, la duplicación de las rutinas sólo se traduce en una mayor ocupación del procedimiento en el disco, normalmente limitada a pocos kbytes. No obstante esto, en algunos casos puede ser útil un ahorro de espacio también para valores tan bajos; por ejemplo, con los diskettes de 5"1/4, que permiten una densidad simple menor de 160 kbytes, los programas pueden incidir por más del 20%. En estos casos existen técnicas de segmentación basadas en la reubicación en áreas de memoria bien definidas, que permiten utilizar algunas partes de procedimiento diferentes.

DIAGRAMA DE FLUJO DEL MENU PRINCIPAL



ESQUEMA DE CONCATENADO MENU PRINCIPAL / MENU SECUNDARIO/PROCEDIMIENTO

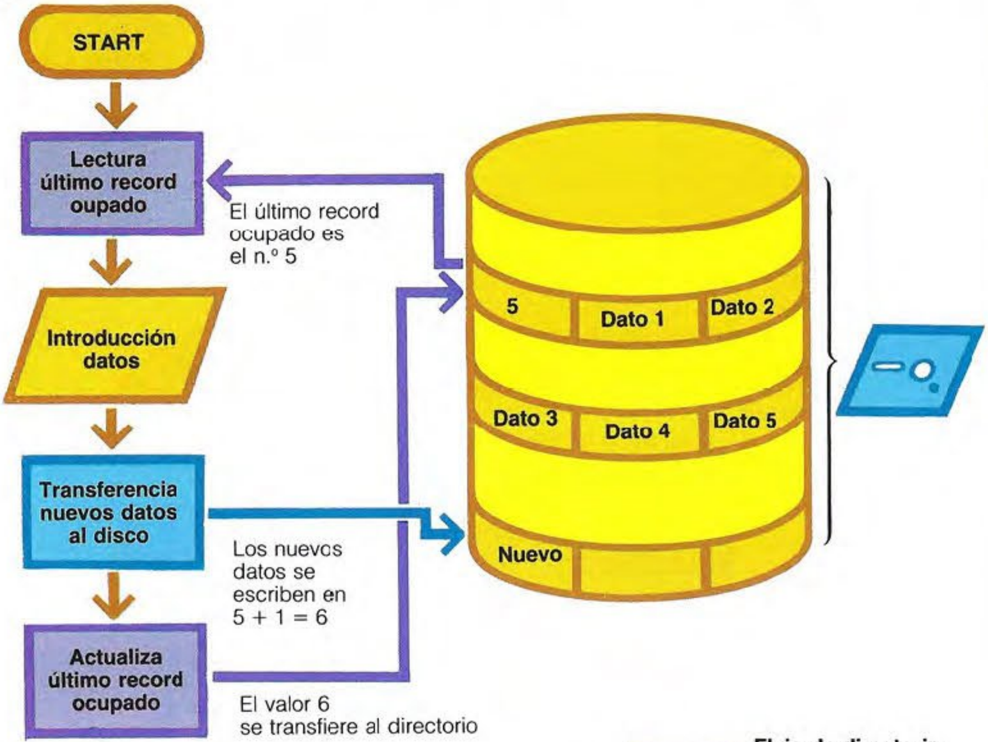


Gestión de los archivos

Cualquiera que sea el problema examinado debe gestionar archivos de datos. Las funciones a realizar son, normalmente, la introducción, la actualización y la impresión. En la página siguiente se han esquematizado los métodos utilizados para la introducción y para la actualización, mientras que en las págs. 1279, 1280 y 1281 se han representado los diagramas de flujo para la gestión de los datos de clientes y de proveedores. Para los otros files, las únicas variaciones corresponden al nombre y al formato del record. Los diagramas de flujo son útiles para los lenguajes poco especializados; para el Cobol, el acceso a los files es tan sencillo que no necesita ninguna indicación. En la pág. 1279 se ha representado un ejemplo en el cual puede verse cómo el diagrama de flujo no dice nada más de lo que pueda decirse con una simple indicación escrita. En el diagrama de flujo correspondiente a la versión Basic se ha introducido un ejemplo de generalización. A la entrada de la rutina se realiza un test sobre un flag para determinar de qué file se trata (Clientes o Proveedores); como los dos formatos de record son idénticos, la rutina puede utilizarse para los dos files simplemente variando el nombre del file a seleccionar.

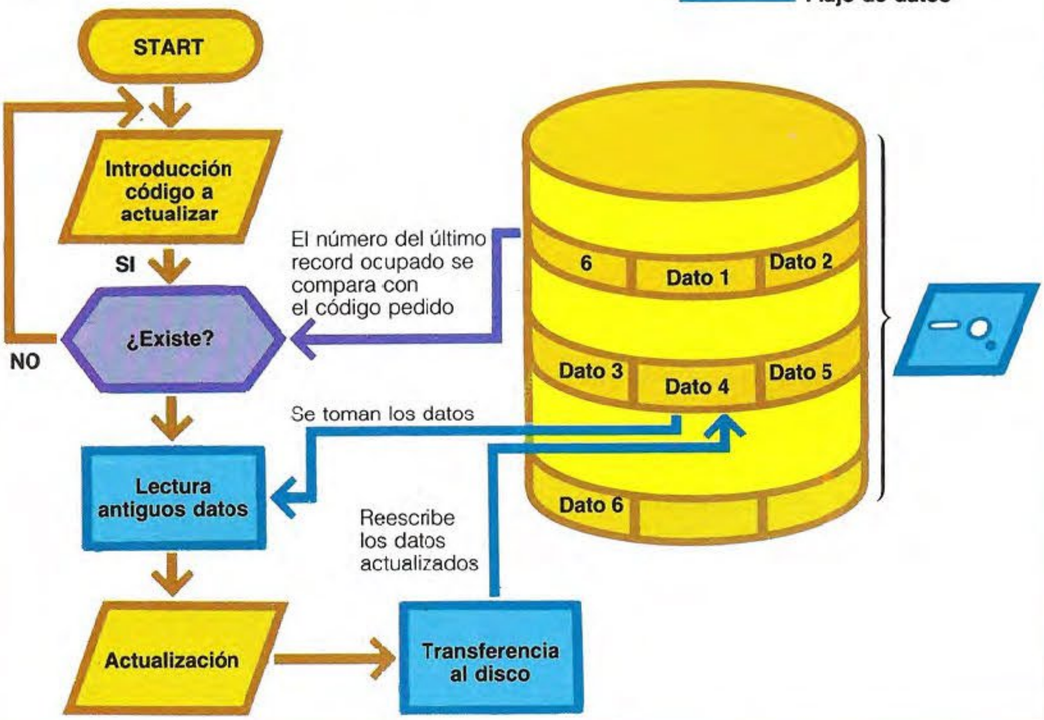
ESQUEMAS PARA LA INTRODUCCION Y LA ACTUALIZACION DE LOS DATOS

Introducción

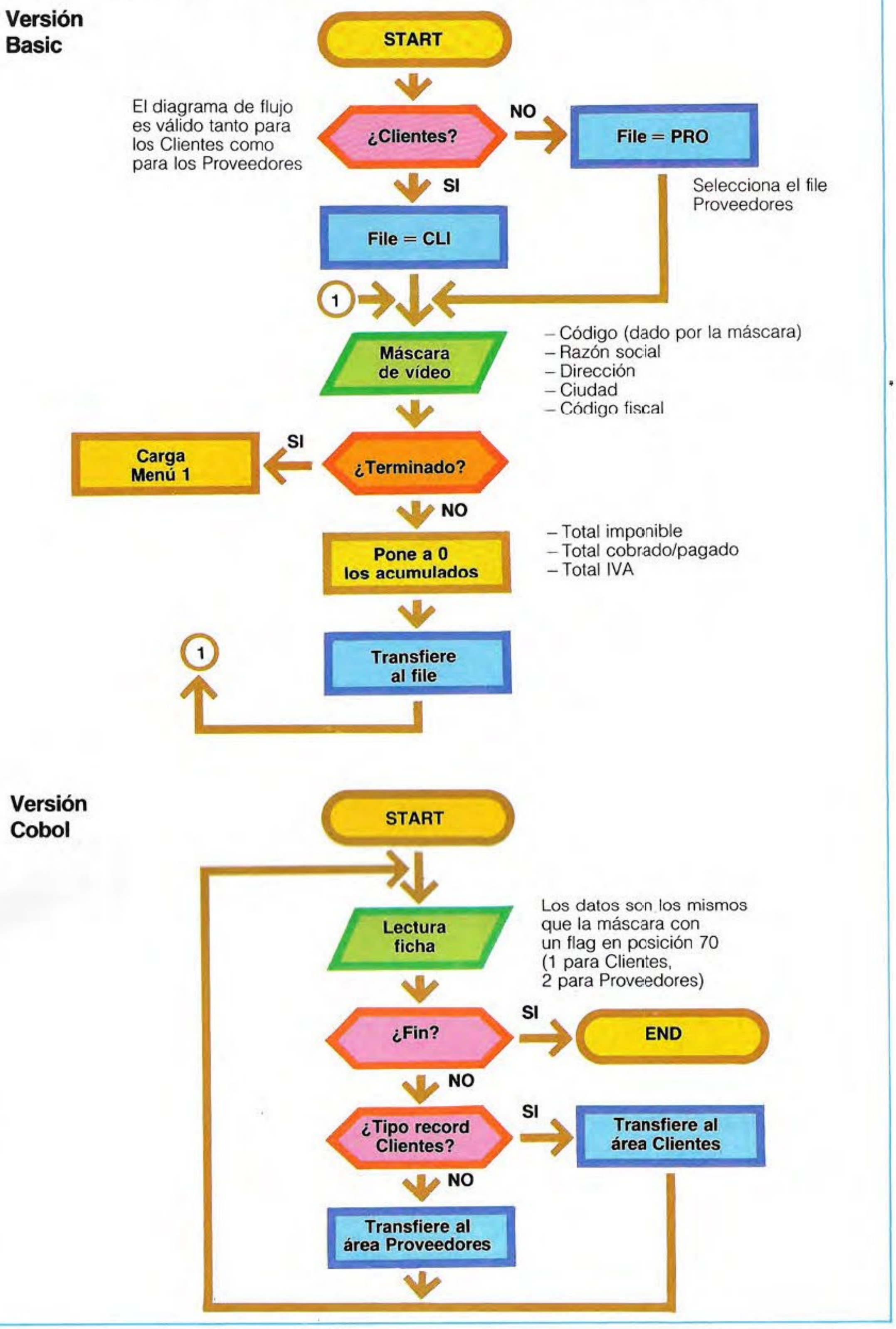




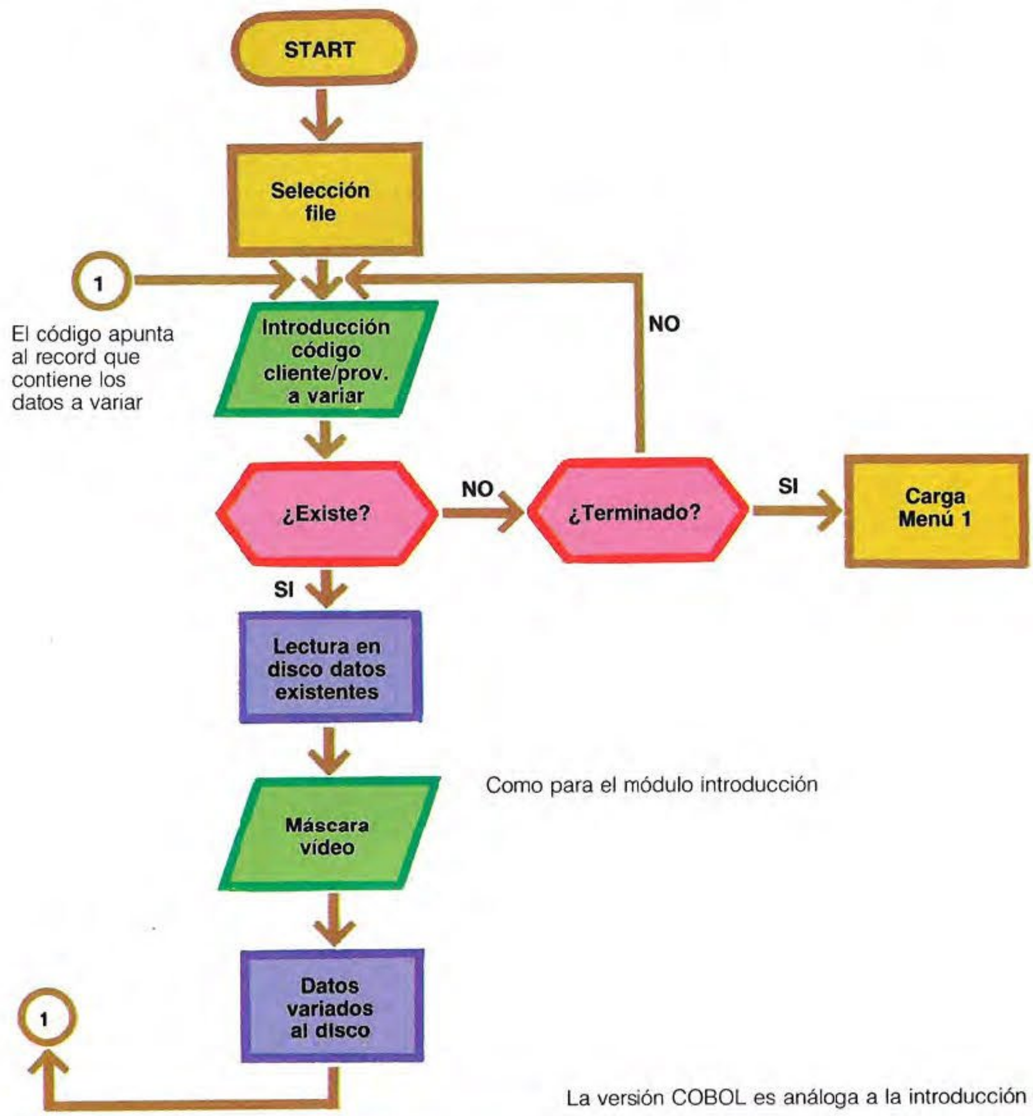
Actualización



INTRODUCCION NUEVOS CLIENTES O PROVEEDORES



VARIACIONES SOBRE LOS FILES CLIENTES Y PROVEEDORES

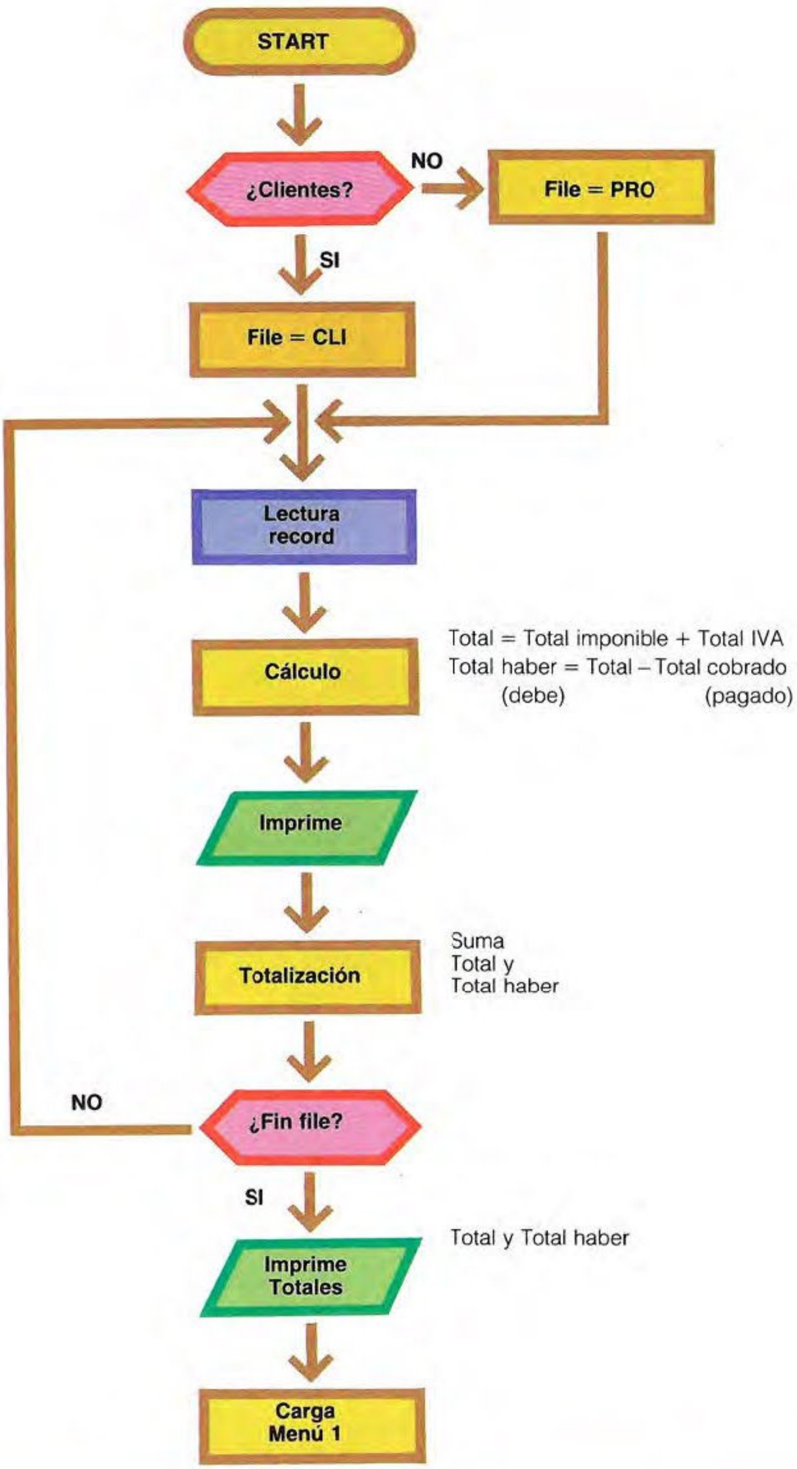


Una ulterior parametrización puede obtenerse transfiriendo como parámetro también el formato del record. De este modo, una sola rutina puede utilizarse para todas las aplicaciones. En la pág. 1282 se ha representado el diagrama de flujo de una rutina generalizada para la lectura o la escritura de un record. Los parámetros a transferir son

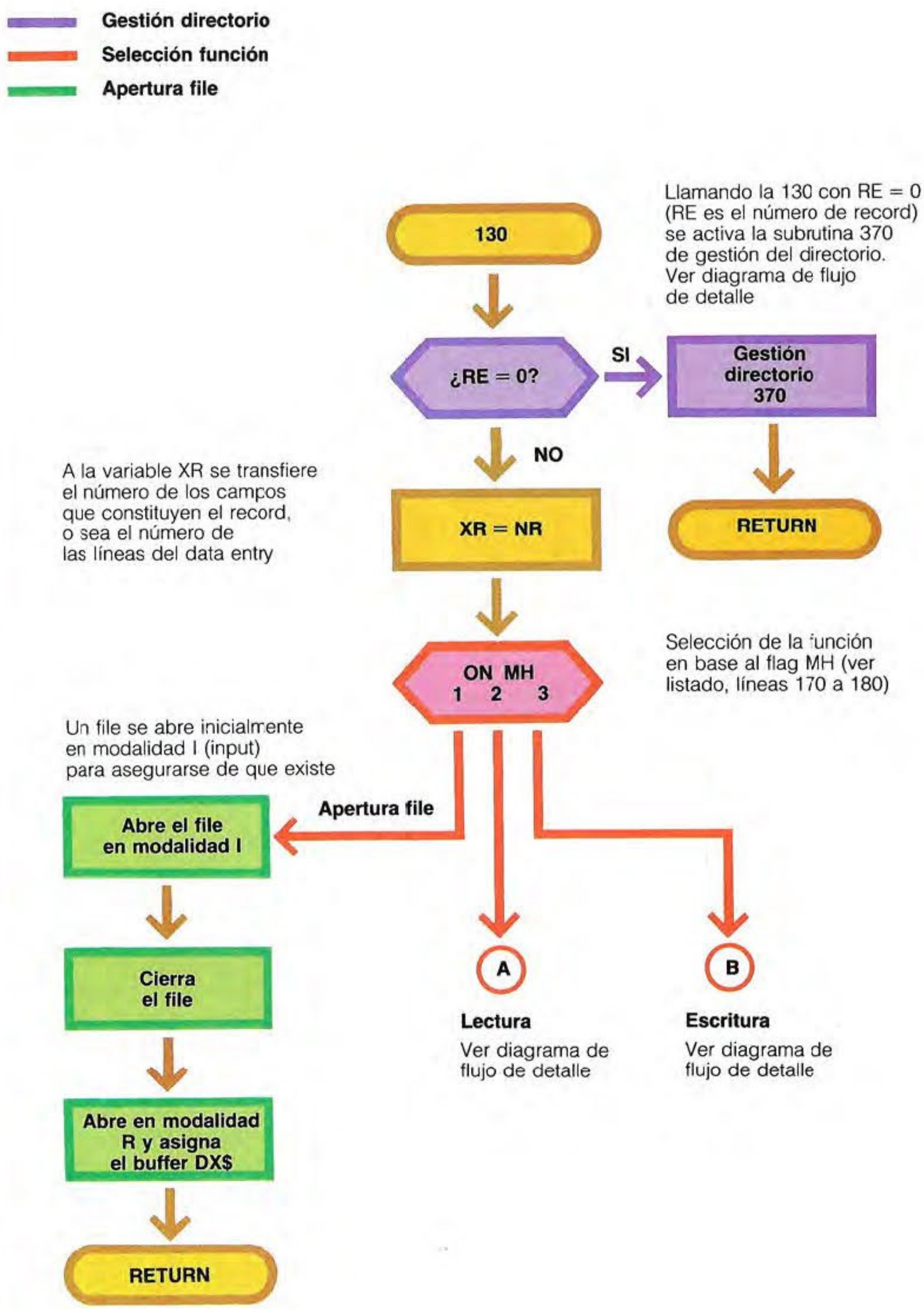
— Nombre del file, en la variable NM$
— Número de los campos, en la variable NR
— Longitud de cada campo, en bytes, definida como diferencia entre los valores de la matriz PD (bytes iniciales de cada campo) y PA (bytes finales). Por ejemplo, si el primer campo de datos tiene PD (1) = 1 y PA (1) = 30, la longitud en bytes del campo es N = PA (1) – PD (1) + 1 = 30 – 1 + 1 = 30 bytes.
— RE, número del record a leer o a escribir

En respuesta, la rutina proporciona el contenido del record en la matriz BF$, en el que cada campo está separado de los demás; por ejem-

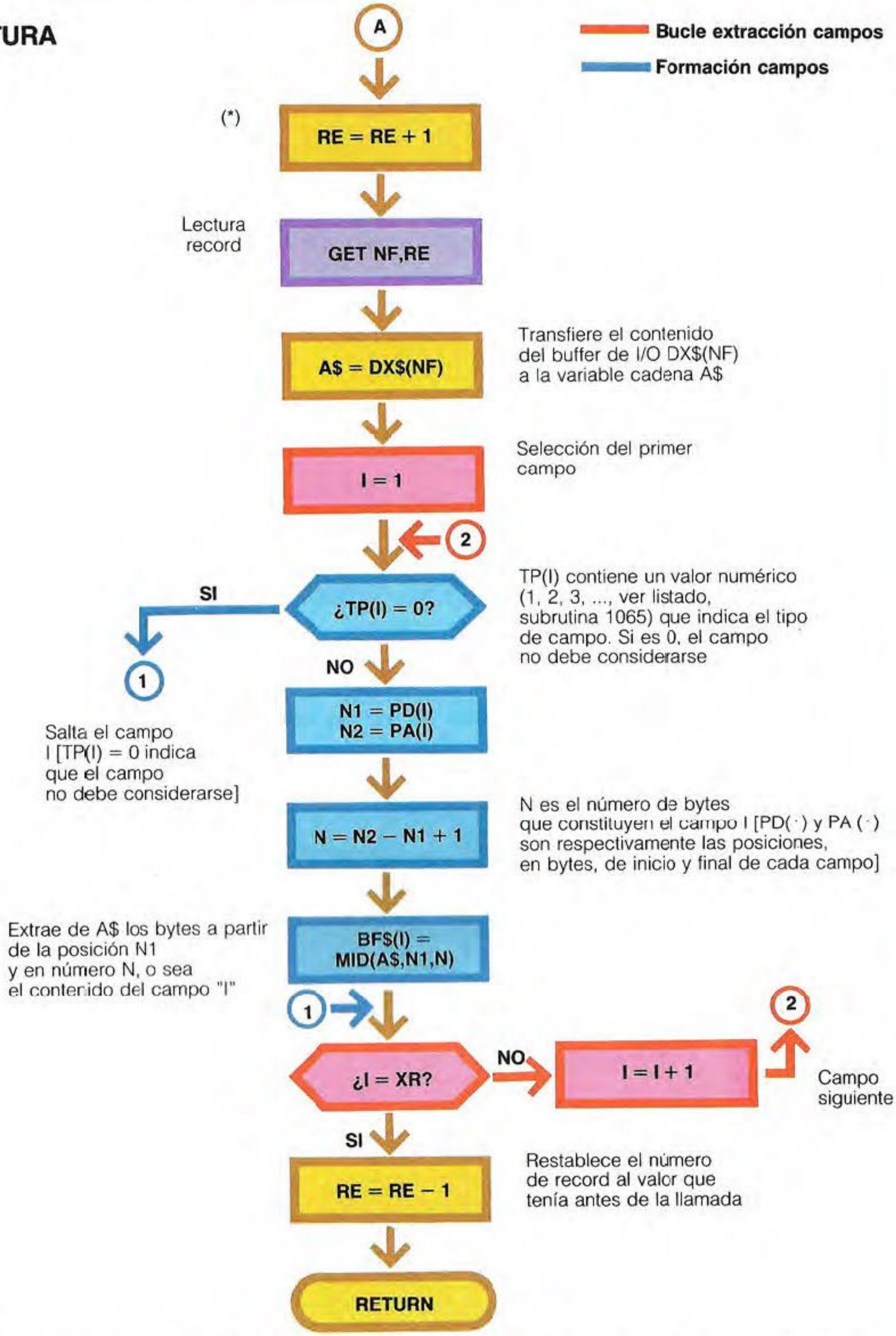
SITUACION CLIENTES/PROVEEDORES



SUBRUTINA DE LECTURA/ESCRITURA DISCO PARAMETRIZADA

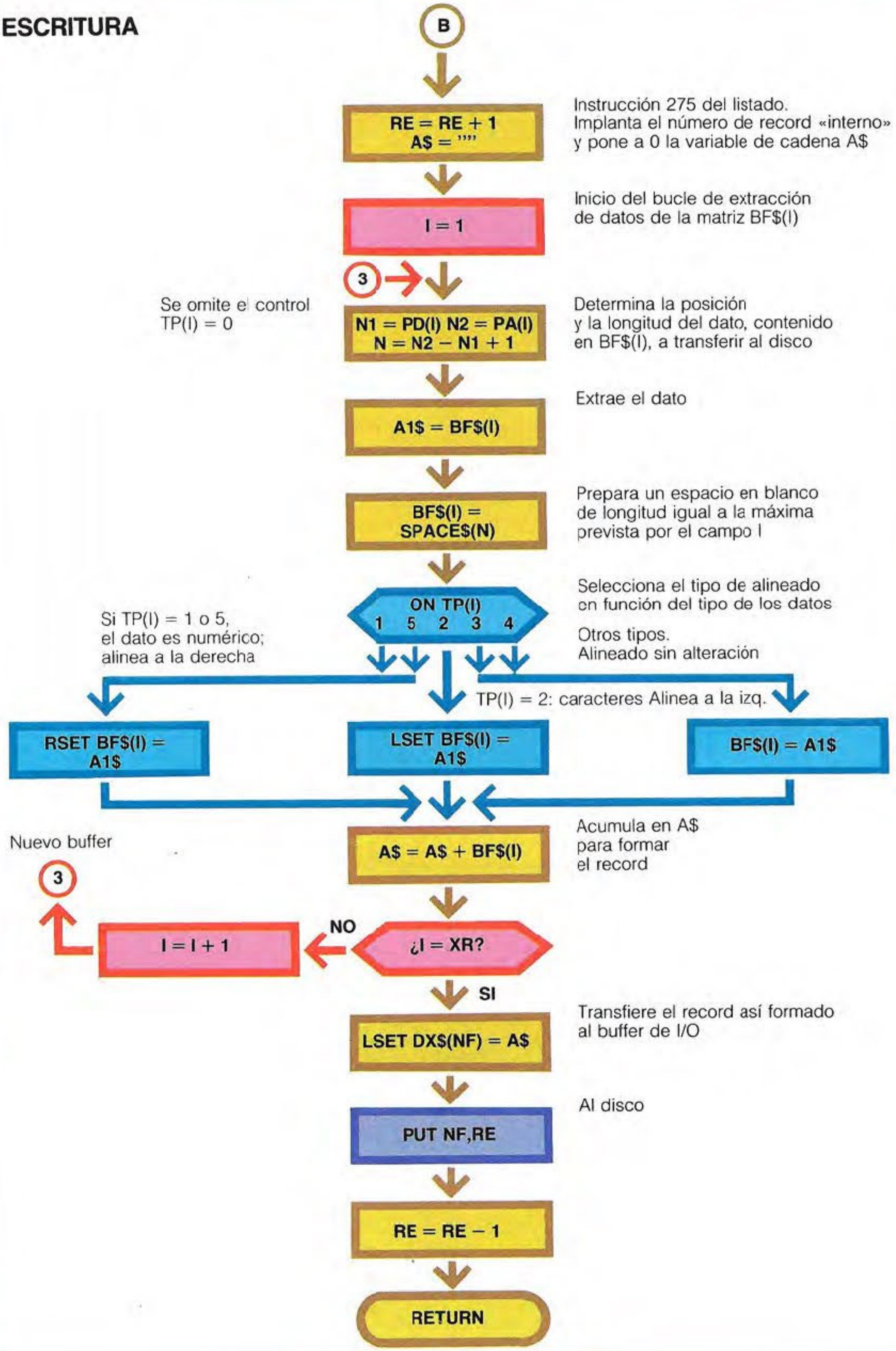


LECTURA

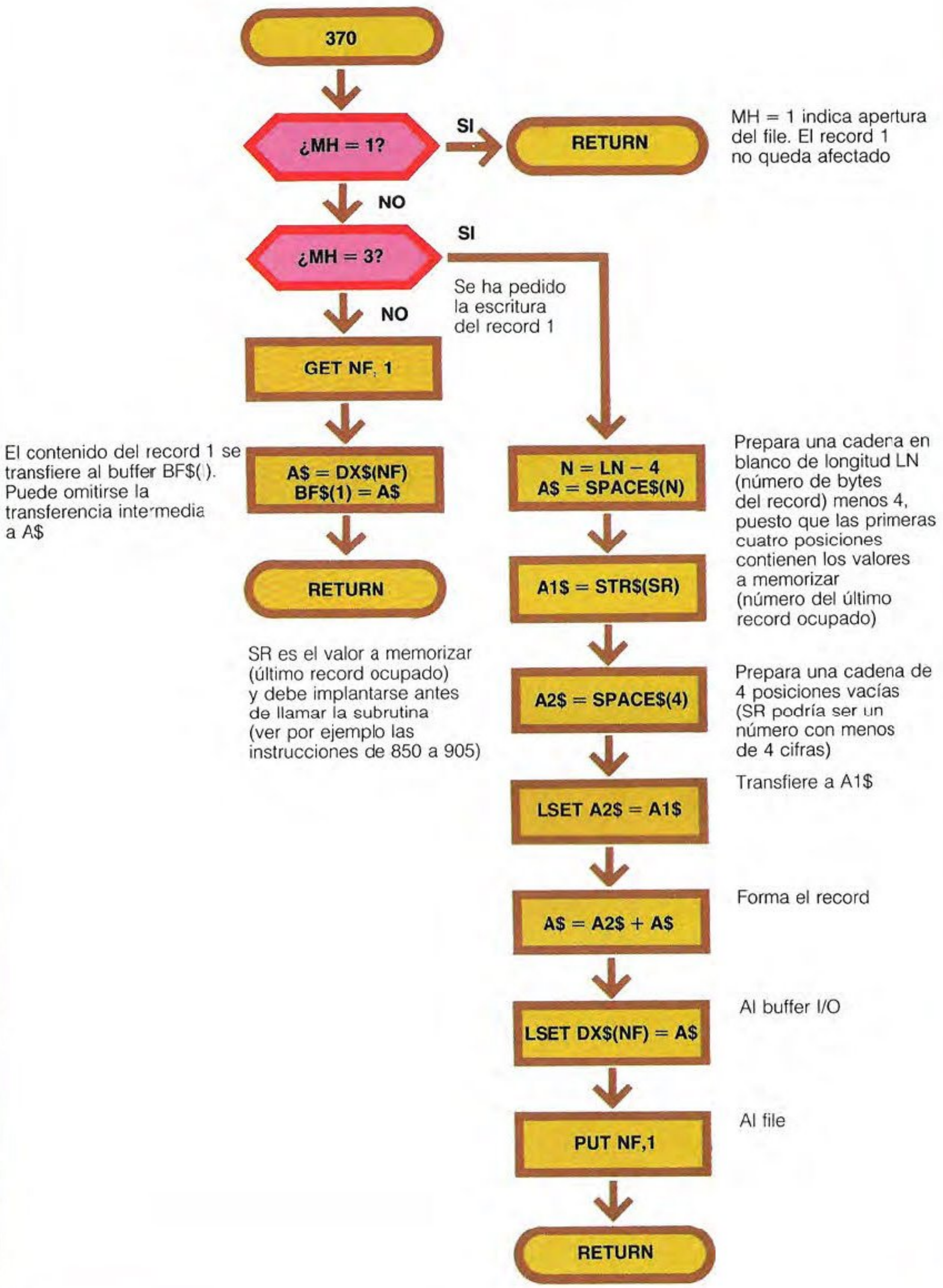


(*) Como el record número 1 se utiliza para el directorio, los datos se memorizan a partir del record 2: el primer dato está en posición 2, el segundo en posición 3, etc.
Para eliminar esta diferencia, la rutina escribe los datos en posición + 1 respecto a su sucesión ordinal de introducción. De esta manera, el programa llamador pide escribir el número de dato en posición 1 y la rutina lo traslada a 2. Esta necesidad desaparece en el ámbito de los Sistemas Operativos que numeran los records a partir de 0.

ESCRITURA


B
RE = RE + 1
A$ = ""
Instrucción 275 del listado.
Implanta el número de record «interno»
y pone a 0 la variable de cadena A$
I = 1
Inicio del bucle de extracción
de datos de la matriz BF$(I)
3
Se omite el control
TP(I) = 0
N1 = PD(I) N2 = PA(I)
N = N2 − N1 + 1
Determina la posición
y la longitud del dato, contenido
en BF$(I), a transferir al disco
A1$ = BF$(I)
Extrae el dato
BF$(I) =
SPACE$(N)
Prepara un espacio en blanco
de longitud igual a la máxima
prevista por el campo I
ON TP(I)
1 5 2 3 4
Selecciona el tipo de alineado
en función del tipo de los datos
Si TP(I) = 1 o 5,
el dato es numérico;
alinea a la derecha
Otros tipos.
Alineado sin alteración
TP(I) = 2: caracteres Alinea a la izq.
RSET BF$(I) =
A1$
LSET BF$(I) =
A1$
BF$(I) = A1$
Nuevo buffer
3
A$ = A$ + BF$(I)
Acumula en A$
para formar
el record
I = I + 1
NO
¿I = XR?
SI
LSET DX$(NF) = A$
Transfiere el record así formado
al buffer de I/O
PUT NF,RE
Al disco
RE = RE − 1
RETURN

GESTION DEL DIRECTORIO



plo, BF$ (1) contiene el primer campo, BF$ (2) el segundo, y así sucesivamente. En la fase de escritura en disco, la matriz BF$ debe prepararse con los datos a transferir y los otros parámetros quedan sin variación. La selección entre la lectura y la escritura se produce con el flag MH (MH = 1 para la lectura, MH = 2 para la escritura). Aquí al lado y en las páginas sucesivas (instrucciones de la línea 130 a la 465) se ha representado el listado de la subrutina en la versión Basic 80 bajo CP/M. La subrutina está incluida en un programa de ejemplo que muestra su uso. Ésta prevé la gestión separada del primer record del file (directorio) en el que se ha memorizado el número del último record ocupado por los datos. Para activar esta función debe llamarse la rutina poniendo RE = 0 para determinar su posición en el interior del file; a continuación, los datos se transfieren al file en la posición (record) que se obtiene sumando 1 al contenido del directorio, y al final se actualiza el directorio. Esta metodología se ilustra en las líneas 800 a 905, que constituyen la subrutina de introducción de datos. Las versiones en ambiente diferente del CP/M sólo difieren en las instrucciones estrictamente inherentes a la gestión del disco, y las traducciones a otras versiones del Basic sólo requieren la sustitución de estas instrucciones. La única limitación se debe a la necesidad de memorizar también los valores numéricos en forma de cadena; esto implica una transformación de un número en cadena en la fase de escritura, y la transformación inversa (de cadena a número) después de la lectura. Estas funciones no están previstas en el listado y deben activarse, repetitivamente, antes y después de la rutina.

Emisión de las facturas

La emisión de una factura es un acto regulado por la legislación fiscal y, por tanto, debe controlarse y guiarse cuidadosamente. En la pág. 1292 se ha representado el diagrama de flujo de principio de la rutina, en la que se presentan algunos de los principales controles.

El principal control a efectuar es sobre la fecha. En el file DIR se memorizan la fecha y el número de la última factura emitida. Activando el procedimiento, el propio programa puede proporcionar los números de factura acumulados y realizar además un control de congruencia sobre la fecha: la fecha introducida por consola debe ser igual o posterior a la última fecha memorizada.

Empleo del ordenador en el National Hurricane Meteo Center.



Marka-Leo de Wys

PROCEDIMIENTO DE FACTURACION. GESTION ARCHIVOS

```
10 OPTION BASE 1
15 '    FILE = BASIC
20 '
25 '    VERSION : 23-07-84 : CP/M - BASIC 80
30 DEFINT A-V
35 '    Las variables utilizadas para cálculos deben empezar con Z:
40 DEFDBL Z
45 DIM TF (20)                              ' Teclas funcionales
50 FOR I=1 TO 20: READ TF(I):NEXT I         ' Lectura valores teclas funcionales
55 DATA 10!,8!,12!,11!,16!,17!,13!,22!,18!,0!,19!,20!,21!,0!,0!,0!,0!,0!,0!,0!
60 DIM DX$(4)
65 DIM TP(20),PA(20),PD(20),LD(20)
70 DIM D$(20),B$(20),BF$(20),AT$(20),BR$(20),CN$(20)
75 DIM IS(5000)
80 '
85 '
90 '
95 '    *  FUNCIONES  *
100 '
105 BI$=CHR$(27)+"+"+"CHR$(7)               ' Borra la pantalla
110 PRINT BI$
115 '
120 GOSUB 470                               ' Menú de las funciones previstas
125 STOP
130 '
135 '   ** GESTION DATOS
140 '     FILE     GDAT     CONTROLADO 23-07-84
145 ' Record = RE Tipo record = TR BF$(20)
150 ' Buffer de 1 a 4      DX$(4)  = 4 Files
155 ' NF = Número del File = Número del Buffer
160 ' NL = Número de líneas
165 ' MH =
170 '         1 Apertura
175 '         2 Lectura
180 '         3 Escritura
185 IF RE=0 THEN GOSUB 370:RETURN         ' RE=0 para gestionar el directorio del file
190 XL=NL                                 ' NUMERO DE CAMPOS (1310)
195 ON MH GOTO 200,220,275
200 ' APERTURA
205 OPEN "i", NF,NM$    ' Abre en modo I para asegurarse de que exista el file
210 CLOSE NF:OPEN "r", NF,NM$,LN:FIELD NF,LN AS DX$(NF):RETURN
215 ' La linea anterior abre en modo R y asigna el Buffer de I/O
220 ' LECTURA
225 RE=RE+1        ' Ver diagrama flujo
230 GET NF,RE:A$=DX$(NF)       ' Lectura y transferencia de los datos a A$
235 ' *Forma los buffer BF$(,) en función de las longitudes
240 '   de los campos definidos en la subrutina 1310
245 FOR I=1 TO XL:IF TP(I)=0 GOTO 255
250 N1=PD(I):N2=PA(I):N=N2-N1+1:BF$(I)=MID$(A$,N1,N)
255 NEXT I
260 RE=RE-1        ' Ver diagrama flujo
265 RETURN
270 '
275 ' ESCRITURA
280 RE=RE+1
285 A$=""
290 FOR I=1 TO XL  : IF TP(I)=0 GOTO 345        ' Selección de uno de los campos
295 N1=PD(I):N2=PA(I): N=N2-N1+1                ' Cálculo de la longitud en bytes
300 A1$=BF$(I)                                  ' Salva el contenido
305 BF$(I)=SPACE$ (N)                           ' Prepara el espacio necesario
310 '                                             en base a la longitud del campo
315 ON TP(I) GOTO 320,325,330,330,320           ' Selecciona en base al tipo
320 RSET BF$(I)=A1$:GOTO 335                    ' Numérico a la derecha
325 LSET BF$(I)=A1$:GOTO 335                    ' Caracteres a la izquierda
```

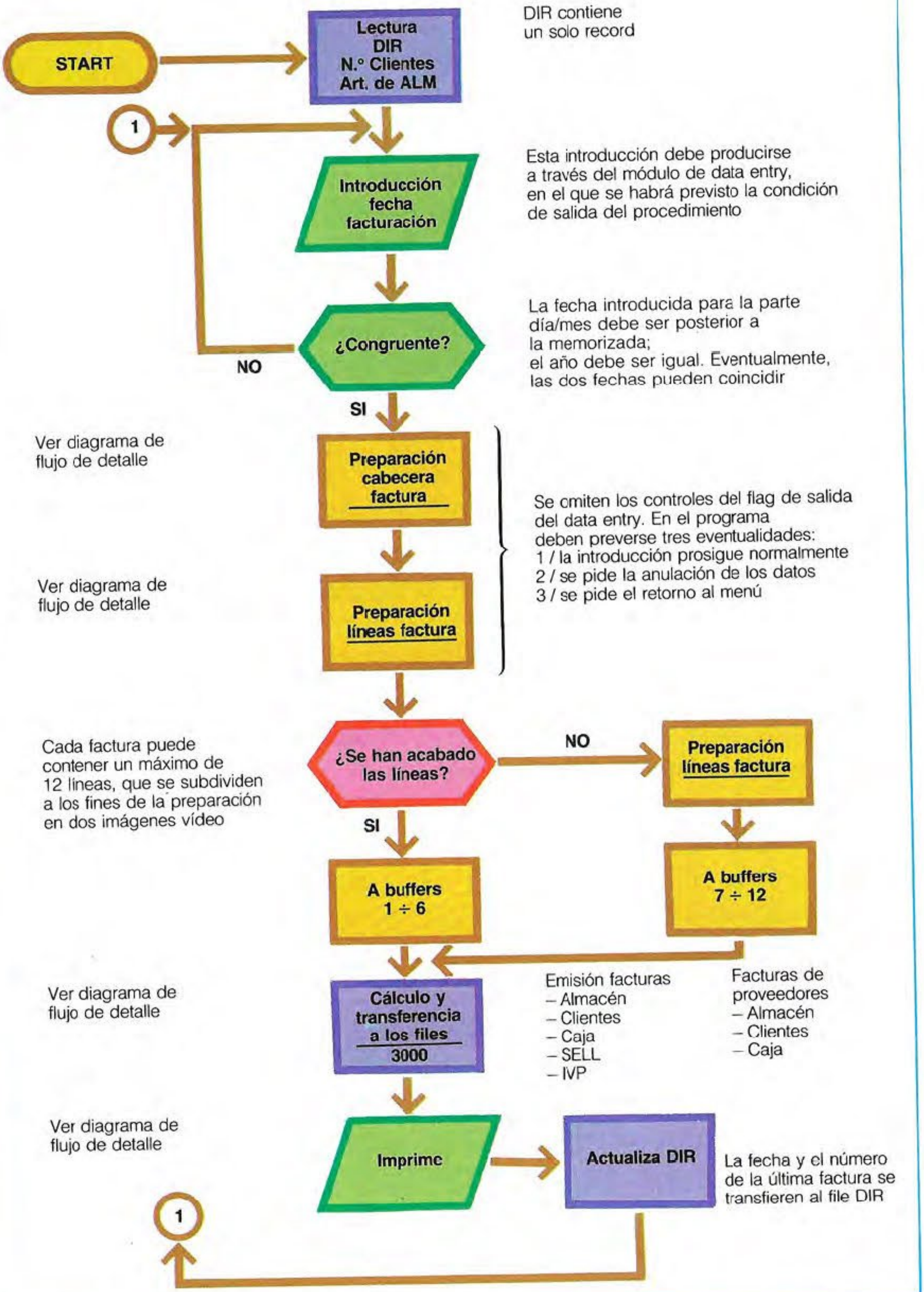
```
330 BF$(I)=A1$                                ' Otros tipos no variados
335 A$=A$+BF$(I)                              ' Suma los buffer
340 '                                           en una sola línea
345 NEXT I
350 LSET DX$(NF)=A$                           ' Transfiere al buffer I/O
355 PUT NF ,RE                                ' Escribe el record RE en el file NF
360 RE=RE-1
365 RETURN
370 ' Gestión record 1
375 '
380 IF MH=1 THEN RETURN                       ' MH=1 para sólo apertura del file
385 IF MH=3 GOTO 410                          ' A la escritura record 1
390 ' Lectura
395 GET NF,1                                  ' Lectura record 1
400 A$=DX$(NF): BF$(1)=A$                     ' Transfiere al buffer 1
405 RETURN
410 ' Escritura
415 N=LN-4:A$=SPACE$(N)                        ' Prepara una cadena vacía
420 '                                            de longitud LN-4
425 A1$=STR$(SR)                               ' Convierte el número SR
430 '                                            en caracteres
435 A2$=SPACE$(4):LSET A2$=A1$                 ' Lo transfiere a
440 '                                            un campo de longitud 4
445 A$=A2$+A$                                  ' Suma la cadena vacía (415)
450 '                                            para tener un record de longitud LN
455 LSET DX$(NF)=A$                            ' Transfiere al buffer I/O
460 PUT NF,1                                   ' Al file NF Record = 1
465 RETURN
470 '
475 '   MENU1
480 '
485 '   VERS.:27-07-84
490 '   TECLAS FUNCIONALES SON HABILITADAS POR MENU PRINCIPAL - FILE : MENU
495 GOSUB 575
500 ON V GOTO 505, 510, 525, 525, 525, 525, 525, 525, 525, 525
505 GOSUB 775:GOTO 485
510 GOSUB 910:GOTO 485
515 '   Las voces del MENU activadas son 1 y 2
520 '   Las otras se ignoran
525 GOTO 485
530 PRINT BI$
535 CK=5:X=30:Y=10:GOSUB 755
540 PRINT CHR$ (27)+"G1"
545 PRINT CHR$ (27)+"G0"
550 RETURN
555 END
560 '   ------------------------------------------------------------
565 '   * SELECCION DE LA FUNCION *
570 '   ------------------------------------------------------------
575 PRINT BI$
580 CK=5:X=2:Y=1:GOSUB 755:PRINT "E.G.S. Gestión archivo clientes",
585 CK=5:X=15:Y=3:GOSUB 755                 ' Posiciona el cursor en X,Y
590 PRINT "1-INTRODUCCION"
595 Y=Y+1:GOSUB 755
600 PRINT "2-VARIACIONES"
605 Y=Y+1:GOSUB 755
610 PRINT "3- ................."
615 Y=Y+1:GOSUB 755
620 PRINT "4- ................."
625 Y=Y+1:GOSUB 755
630 PRINT "5- ................."
635 Y=Y+1:GOSUB 755
640 PRINT "6- ................."
645 Y=Y+1:GOSUB 755
```

```
650 PRINT "7- .................."
655 Y=Y+1:GOSUB 755
660 PRINT "8- .................."
665 Y=Y+1:GOSUB 755
670 PRINT "9- .................."
675 Y=Y+1:GOSUB 755
680 PRINT "0- FIN TAREA"
685 X=1:Y=Y+2:GOSUB 755
690 PRINT "INTRODUCIR OPCION ELEGIDA"
695 X=26:GOSUB 755:A$=INPUTS(1):PRINT A$
700 V=VAL(A$)
705 IF V>9 GOTO 730
710 PRINT BI$
715 IF V=0 THEN V=10
720 RETURN
725 '  ** ERROR **
730 PRINT CHR$(7), CHR$(7)
735 X=42:GOSUB 755
740 PRINT "NO VALIDO"
745 GOTO 695
750 '
755 '   DESPLAZAMIENTO CURSOR
760 '     CURS-X/Y
765 '    La instrucción que sigue depende del tipo de máscara usada
770 PRINT CHR$(27)+CHR$(61)+CHR$(31+Y)+CHR$(31+X);
775 RETURN
780 '  ** FIN MENU **
785 '  *
790 '  *
795 '  *** INTRODUCCION ***
800 ' APERTURA
805 RE=1
810 NF=1:GOSUB 1305                      ' Prepara formato
815 MH=1:GOSUB 130                       ' Abre el file
820 ' Lectura del último record ocupado
825 RE=0:MH=2
830 GOSUB 130
835 RE=VAL(BF$(1))                       ' RE = primer record libre+1
840 X=1:Y=1:GOSUB 1280                   ' Posiciona el cursor
845 PRINT "CODIGO CLIENTE : ";RE         ' Presenta el número de código
850 '                                      (record) del siguiente cliente
855 GOSUB 1305                           ' Prepara formato
860 GOSUB 1065                           ' Presenta en video
865 GOSUB 1250                           ' Data Entry
870 ON F1 GOTO 875,880,840
875 RETURN                               ' Retorna (los datos son anulados)
880 MH=3:GOSUB 130                       ' Introducción nuevo cliente
885 SR=RE
890 RE=0:GOSUB 130                       ' Actualiza el directorio (record 1)
895 GOTO 820                             ' Nueva introducción
900 CLOSE NF
905  RETURN
910 ' Variaciones
915 RE=1
920 NF=1:GOSUB 1305                      ' Prepara formato
925 MH=1:GOSUB 130                       ' Abre el file
930 RE=0:MH=2:GOSUB 130                  ' Lectura directorio
935 MX=VAL(BF$(1))                       ' MX = último código cliente utilizado
940 X=1:Y=1:GOSUB 1280
945 PRINT" CODIGO CLIENTE :";
950 KC=20:MD=0:KR=1:TP(1)=1              ' *** Prepara formato para introducción
955 '                                          código del cliente a variar
960 NF=1:PD(1)=1:PA(1)=3:LN=3:LD(1)=3
965 GOSUB 1250                           ' Data Entry para la lectura del código
```

```
970 ON F1 GOTO 975,980,940
975 RETURN                                 ' Retorno al MENU
980 V=VAL(BF$(1)                           ' Introducción
985 IF V>MX THEN PRINT "ERROR ": GOTO 940          ' Controla que no se haya
990 '                                                 pedido un código mayor
995 '                                                 que el último memorizado
1000 NF=1:GOSUB 1305
1005 RE=V
1010 MH=2:GOSUB 130
1015 FOR I=1 TO NR:TP(I)=4:NEXT I         ' Transforma los campos presentados
1020 GOSUB 1065                           ' Presenta en video
1025 GOSUB 1250                           ' Data Entry para las variaciones
1030 ON F1 GOTO 1035,1040,940
1035 RETURN
1040 MH=3:GOSUB 130
1045 GOTO 940
1050 '
1055 RETURN
1060 '
1065 '
1070 '      **** FORMATOS VIDEO ****
1075 '      VIDE
1080 ' VERSION : 24-07-84
1085 '
1090 ' TIP =
1095               1-NUMERICO
1100 '             2-ALFANUMERICO
1105 '             3-EN SOLO PRESENTACION (Introducción Impedida)
1110 '             4-PRESENTACION CON INTRODUCCION
1115 '             5-CAMPO RESULTADO
1120 '** ENTRADA          NF = número del File **
1125 MD=0
1130 '  Calcula la máxima longitud de las descripciones
1135 FOR I=1 TO NL
1140 IF LEN(D$(I))>=MD  THEN MD =LEN(D$(I))
1145 NEXT I
1150 ' Prepara los Buffer
1155 FOR I=1 TO NL:N=MD-LEN(D$(I))
1160 IF N=0 THEN 1170
1165 D$(I)=D$(I)+SPACE$(N)
1170 NEXT I
1175 FOR I=1 TO NL:IF TP(I)=0 THEN 1200
1180 IF TP(I)=3 OR TP(I)=4 THEN D$(I)=D$(I)+" "+BF$(I):GOTO 1200
1185 N=PA(I)-PD(I)+1
1190 D$(I)=D$(I)+" "+STRING$(N,46)
1195 BF$(I)=SPACE$(N)
1200 NEXT I
1205 '.
1210 ' Escribe en video
1215 X=KC:Y=KL
1220 FOR I=1 TO NL
1225 GOSUB 1280
1230 PRINT D$(I)
1235 Y=Y+1
1240 NEXT I
1245 RETURN
1250 '
1255 ' ***** DATA ENTRY *****
1260 ' ------------------------------------------------
1265 ' El Data Entry es análogo al ya presentado
1270 ' ------------------------------------------------
1275 '
1280 ' Desplazamiento cursor
1285 PRINT CHR$(27)+CHR$(61)+CHR$(31+Y)+CHR$(31+X)
```

```
1290 RETURN
1295 '
1300 RETURN
1305 '
1310 '  FECHA VERS. 23-07-84
1315 ' *** FECHA ***
1320 ' ENTRADA : NF=TIPO RECORD Y NUMERO DEL FILE
1325 '
1330 'SALIDAS :
1335 '            C=POSICION PRIMERA COLUMNA
1340 '            KL=POSICION PRIMERA LINEA PANTALLA VIDEO
1345 '            NL=ÑUMERO DE LINEAS
1350 '            NF=NUMERO DEL FILE
1355 '            D$(*)=DESCRIPCION DE LAS LINEAS
1360 '            TP(*)=TIPO DE CAMPO VISTO : VIDE - 1070
1365 '            PD(*)=PUNTERO INICIO CAMPO
1370 '            PA(*)=PUNTERO FINAL CAMPO
1375 '            LN=NUMERO DE CARACTERES
1380 '            NM$=NOMBRE DEL FILE
1385 ON NF GOTO 1400
1390 '
1395 '  TR=2 : DATOS
1400 KC=2:KL=3:NL=8:N=NL
1405 RESTORE 1415
1410 FOR I=1 TO N:READ D$(I):NEXT I
1415 DATA " 1 - Razón Social"
1420 DATA " 2 - Dirección"
1425 DATA " 3 - Ciudad"
1430 DATA " 4 - Partida I.V.A."
1435 DATA " 5 - Total Facturas"
1440 DATA " 6 - Total Pagado"
1445 DATA " 7 - Total Saldo"
1450 DATA " 8 - ..............."
1455 FOR I=1 TO N:READ TP(I): NEXT I
1460 DATA 2!,2!,2!,1!,5!,5!,5!,2!
1465 FOR I=1 TO N:READ PD(I):NEXT I
1470 DATA 1!,31!,61!,76!,87!,98!,109!,120!
1475 FOR I=1 TO N:READ PA(I):NEXT I
1480 DATA 30!,60!,75!,86!,97!,108!,119!,128!
1485 LN=PA(N):NM$="A:PRO"
1490 FOR I=1 TO N:LD(I)=PA(I)-PD(I)+1:NEXT I
1495 N=NR+1:FOR I=N TO 20:D$(I)=" ":TP(I)=0:PD(I)=0:PA(I)=0:NEXT I
1500 RETURN
1505 '
1510 '  *** CAMPO RESULTADO ***
1515 Z5=VAL(BF$(5))
1520 Z6=VAL(BF$(6))
1525 Z7=Z5-Z6
1530 BF$(7)=SPACE$(11)
1535 A2$=STR$(Z7)
1540 RSET BF$(7)=A2$
1545 RETURN
1550 '
```

EMISION DE FACTURAS



SUBRUTINAS DE CALCULO Y TRANSFERENCIA A LOS FILES

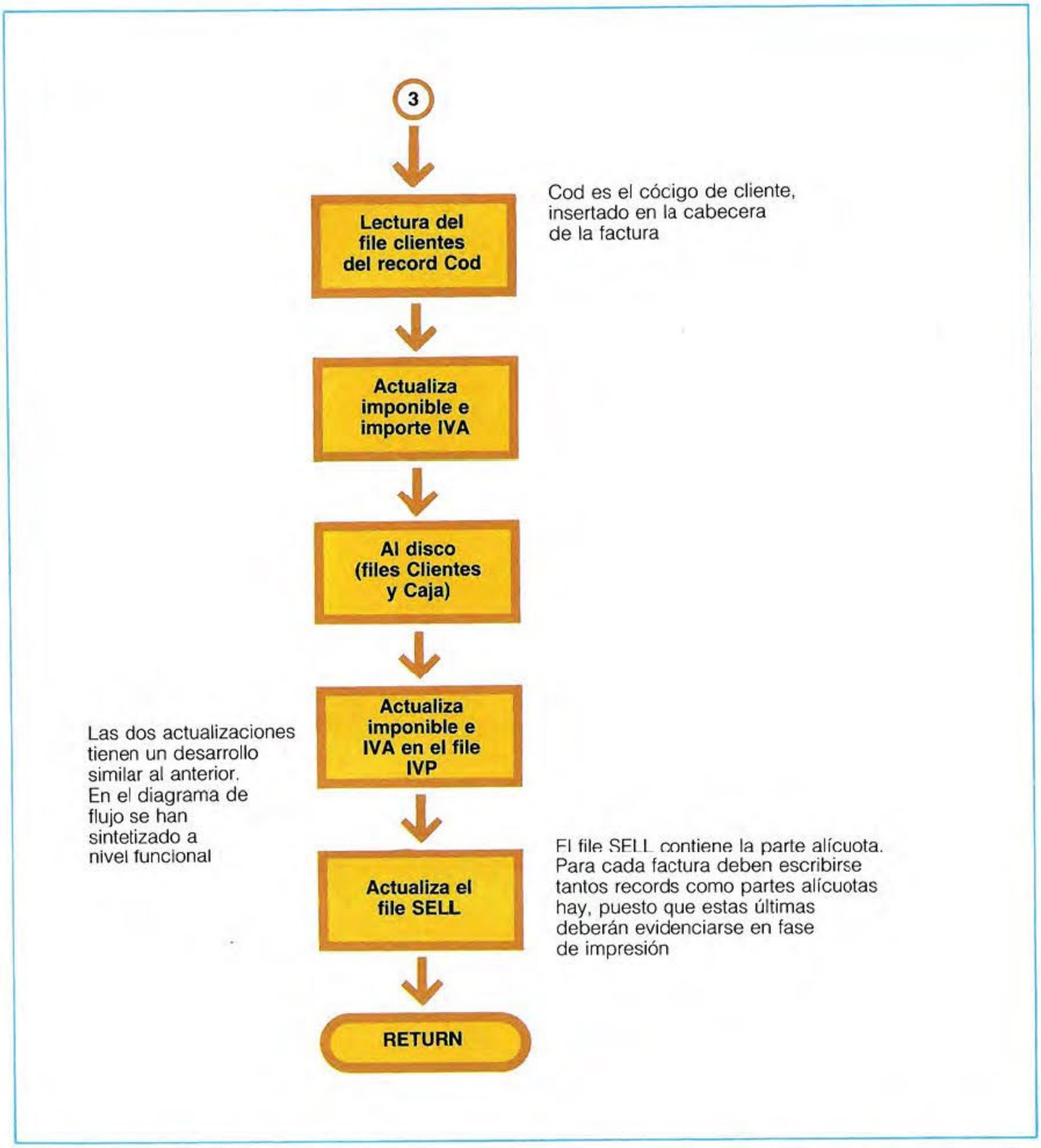


El record de almacén ya se ha leído durante la introducción de las líneas (ver el correspondiente diagrama de flujo)

La actualización de las existencias se realiza sólo al final de la compilación de la factura, para poder variar o anular cada línea sin tener que prever complejos procedimientos de corrección de las existencias

Totalización de los disponibles y de los importes IVA divididos en partes alícuotas

Se ha terminado el proceso de las líneas de una factura



Este tipo de control también está previsto para las versiones Cobol, en el ámbito de las cuales los datos se introducen por fichas, para verificar por lo menos formalmente la correcta perforación de las mismas.

La factura se supone compuesta de dos partes principales: una cabecera que contiene los datos del cliente y una serie de doce líneas sobre las que deberán describirse los movimientos de almacén y de caja.

En el diagrama de flujo representado, las dos funciones de preparación de la cabecera y de las líneas se realizan en dos rutinas diferentes que utilizan un data entry parametrizado. Según la función a ejecutar, los parámetros a transferir al data entry deben colocarse a los correspondientes valores. En el diagrama de flujo de la página anterior se ha representado un trazado del procedimiento a seguir y, arriba, lo que corresponde a la preparación de la cabecera.

El número de líneas de la factura puede impedir su presentación completa. Por este motivo, en la pág. 1292 se ha previsto un método que permite la presentación de 6 líneas cada vez.

Cómo se proyecta un videojuego inteligente (3)

Sobre la base de todo lo que se ha dicho sobre Fred, algunos lectores tenderán a definirlo como un jugador de baja categoría. Les rogamos que sean menos pesimistas en su comparación; en realidad Fred puede jugar bastante bien, por lo menos como un jugador que ya ha acumulado una pequeña experiencia, y puede tender a mejorar sus propias habilidades.
Si para un jugador de carne y hueso esto significa efectuar un buen número de partidas para analizar después su historia en el intento de poner remedio a las propias debilidades, en el caso de Fred será misión del lector ampliar el programa de gestión para aumentar su versatilidad, dado que en el estado en que está no contempla la posibilidad de «aprendizaje».
A través de la implantación de nuevos procedimientos, Fred podrá resolver así algunos problemas de juego más complejos, como «los ojos», las declaraciones de atari, las reglas de ko y otras más. No es difícil realizar y hacer operativos estos procedimientos porque basta una corta lectura de las reglas del Go para identificar su papel en la partida y, en consecuencia, establecer su relación con el programa que gestiona el juego del procesador.
La estructura del módulo JUEGA puede verse en el listado 2. En el interior del módulo JUEGA se ha presentado una llamada a las rutinas definidas EFECTO JUGADA BLANCO, EFECTO JUGADA NEGRO, MODELOS.
Evidentemente, para Fred resulta determinante la posibilidad de analizar las jugadas propias y del adversario: éste realiza un análisis sistemático de las jugadas, y mediante esta operación, condiciona su comportamiento a través de procedimientos automáticos ya preestablecidos.
Los dos módulos EFECTO JUGADA son autodescriptivos; en ellos se realiza la búsqueda de los puntos de libertad, efectuados mediante el reclamo del módulo BUSQUEDA.
Hay la posibilidad de que en esta parte del juego Fred decida efectuar una operación importante, por ejemplo la captura de las fichas de un grupo como prisioneras si éstas no tienen libertad. También se utiliza por primera vez el criterio de evaluación de las elecciones de la jugada a realizar, a través del reclamo del módulo EVALUACION.
Las estructuras de los módulos citados pueden

DESCRIPCION DEL MODULO EFECTO JUGADA BLANCO

```
EFECTO JUGADA BLANCO
  INICIO
    PARA cada cruce x con una ficha negra HAZ
      INICIO
        RECLAMA MAPA (negro, x)
        SI el grupo no tiene libertad
        ENTONCES toma sus fichas
        DE OTRA MANERA
          INICIO
            SI el grupo tiene como máximo una
            libertad
            ENTONCES elige una libertad
            (que no esté en el borde)
            SI el grupo tiene una o dos libertades
            ENTONCES RECLAMA EVALUACION
            para la libertad elegida
          FIN
      FIN
  FIN
```

Este módulo analiza el efecto de la jugada del Blanco con las consecuencias en el escenario del Go Ban. En él se evidencia la interrelación operativa que existe entre los diversos módulos y los criterios que la rigen.

DESCRIPCION DEL MODULO EFECTO JUGADA NEGRO

```
EFECTO JUGADA NEGRO
  INICIO
    PARA cada cruce x con una ficha blanca HAZ
      INICIO
        RECLAMA MAPA (blanco, x)
        SI el grupo tiene exactamente una libertad
        ENTONCES
          INICIO
            Coloca en aquel punto la ficha negra
            Retira la ficha blanca
          FIN
        DE OTRA MANERA
          SI el grupo tiene dos o más libertades
          ENTONCES
            INICIO
              Elige una libertad
              RECLAMA EVALUACION
              para la libertad elegida
          FIN
      FIN
  FIN
```

Como en el anterior, también en este módulo se analiza la jugada de un jugador.
Fred evalúa qué cruces ofrecen las mejores posibilidades para su próxima jugada.

verse en las descripciones representadas en esta página y en la página anterior.
El módulo CONTROL, llamado por la rutina EVALUACION impide que en el Go Ban se repita una situación que ya ha existido.
El procedimiento MAPA lo utilizará Fred para buscar y analizar las libertades de un grupo de fichas en el Go Ban. Cuando un grupo de fichas está conectado, es de fundamental importancia para el adversario saber si y cuándo éste puede ser atacado con el fin de hacerlo prisionero, o bien si es más útil dirigir fichas hacia otros puntos para reforzar la propia posición defensiva.
Podemos imaginar que este módulo haga viajar sobre el Go Ban un «reconocedor» que analiza cruce por cruce cuál es la situación, la caracterice y la memorice como instrumento indispensable para las sucesivas elecciones.
El comportamiento del módulo de búsqueda y contado MAPA es decisivamente mecánico y repetitivo. Éste reclama por sí mismo de forma recursiva tantas veces como cuantas sirven para identificar la situación en el Go Ban. Evidentemente, posee en su interior comandos de control que gobiernan su gestión, permitiéndole desarrollar su propia tarea de manera precisa. En la actual formulación, éste debe ser entendido, al igual que los otros módulos, como propuesta de un modelo de comportamiento de Fred susceptible de cambios por parte del lector.

DESCRIPCION DEL MODULO MODELOS

```
MODELOS
  INICIO
    PARA cada ficha blanca HAZ
      INICIO
        SI hay un modelo en la
          mesa de los ejemplos
          centrado sobre esta ficha blanca
        ENTONCES
          INICIO
            Considera la jugada sugerida x
            RECLAMA EVALUACION (x, 2)
          FIN
      FIN
  FIN
```

El comportamiento de Fred se ha orientado al establecimiento de configuraciones de juego que le son favorables. Es decir, no sólo debe reconocer la situación en el Go Ban, sino que también debe mover en base a una interpretación que sugiera una actitud ofensiva o defensiva. Este principio actúa a través del módulo MODELOS.

DESCRIPCION DEL MODULO EVALUACION

```
EVALUACION (jugada, libertad)
  INICIO
    RECLAMA OPTIMUS (mejor jugada,
    mejor libertad)
    SI libertad <= mejor libertad E
      CONTROL (jugada) >=2
    ENTONCES
      INICIO
        Mejor jugada = jugada
        Mejor libertad = libertad
      FIN
  FIN
```

El módulo EVALUACION, como los módulos CONTROL y MODELOS, pertenece al grupo de instrumentos que Fred utiliza para realizar tareas específicas de análisis. El lector podrá desarrollar posteriormente este grupo insertándole dos módulos orientados a la solución de situaciones de juego particulares, mejorando así la capacidad de juego de Fred.

DESCRIPCION DEL MODULO CONTROL

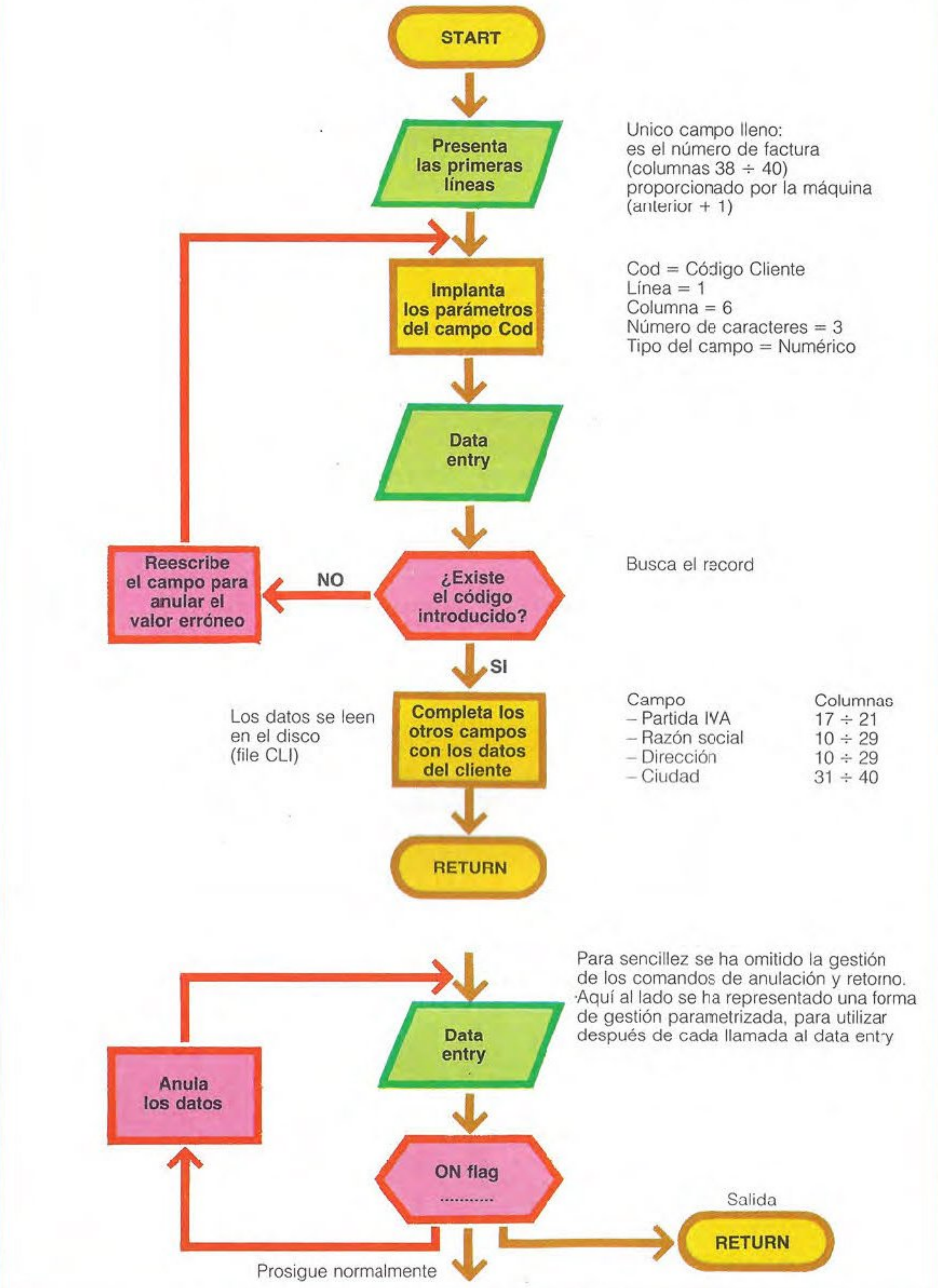
```
CONTROL (jugada)
  INICIO
    Coloca provisionalmente la ficha negra
    RECLAMA MAPA (jugada, negro)
    Retira la ficha negra
    VUELVE al contado de las libertades
  FIN
```

Una de las reglas fundamentales del Go prohíbe expresamente la proposición en el Go Ban de una situación que ya haya existido antes.
Es importante que Fred sepa aplicar correctamente esta regla, llamada ko, en el intento de evitar jugadas suicidas.
CONTROL realiza esta misión reteniendo en la memoria los parámetros actuales de «jugada» y «libertad» antes de que MAPA intervenga para verificar las libertades de aquel punto.

Cuando éste haya comprendido con qué criterio se implanta la gestión de Fred, podrá efectuar experimentalmente variaciones y mejoras en los módulos, a fin de que Fred pueda reaccionar con éxito creciente ante una dinámica de juego más compleja.

Antonio Verga

SUBRUTINA DE PREPARACION DE LA CABECERA



Al final de la introducción, activando una tecla funcional adecuadamente programada, se ordena la transferencia de los datos al disco y la impresión de la factura.
La preparación de las subrutinas que constituyen el procedimiento de la figura de la página anterior no presenta dificultades, excepto para las funciones de introducción de una línea y de impresión. La introducción de una línea de datos para la facturación puede utilizar aún el data entry parametrizado, pero su gestión se hace más compleja. Efectivamente, deben haber varios campos en la misma línea, mientras que el data entry sólo gestiona uno.
Para la impresión pueden presentarse dificultades de alineación, puesto que las facturas deben escribirse sobre módulos predispuestos. Sin embargo, estas dificultades pueden superarse fácilmente con algunas pruebas.
En la página siguiente se ha representado un diagrama de flujo para la gestión de la introducción de una línea de datos. Los campos previstos en introducción son tres: Código artículo, de N caracteres de longitud (la Descripción se presenta automáticamente), Cantidad (el precio se lee en el file almacén), Alícuota IVA (también este campo podría leerse en el file almacén).
En la fase de impresión pueden presentarse otras dificultades con las cantidades expresadas en unidades de medida que admiten cifras decimales (por ejemplo, kg). Durante la introducción de una fila, el operador no puede limitarse a insertar siempre el punto seguido de un número prefijado de decimales; por ejemplo, si el artículo sólo se ha movido en kg es natural omitir los decimales.
Si por comodidad se fija igual a 2 el número de cifras decimales previstas, pueden tenerse los siguientes casos:

- sólo se introducen números enteros: por ejemplo el valor 125
- se introducen valores enteros pero con el punto: 125
- se introducen valores con un decimal: 125.6
- se introducen valores con dos decimales: 125.68
- se introducen valores con más de dos decimales:125.6873

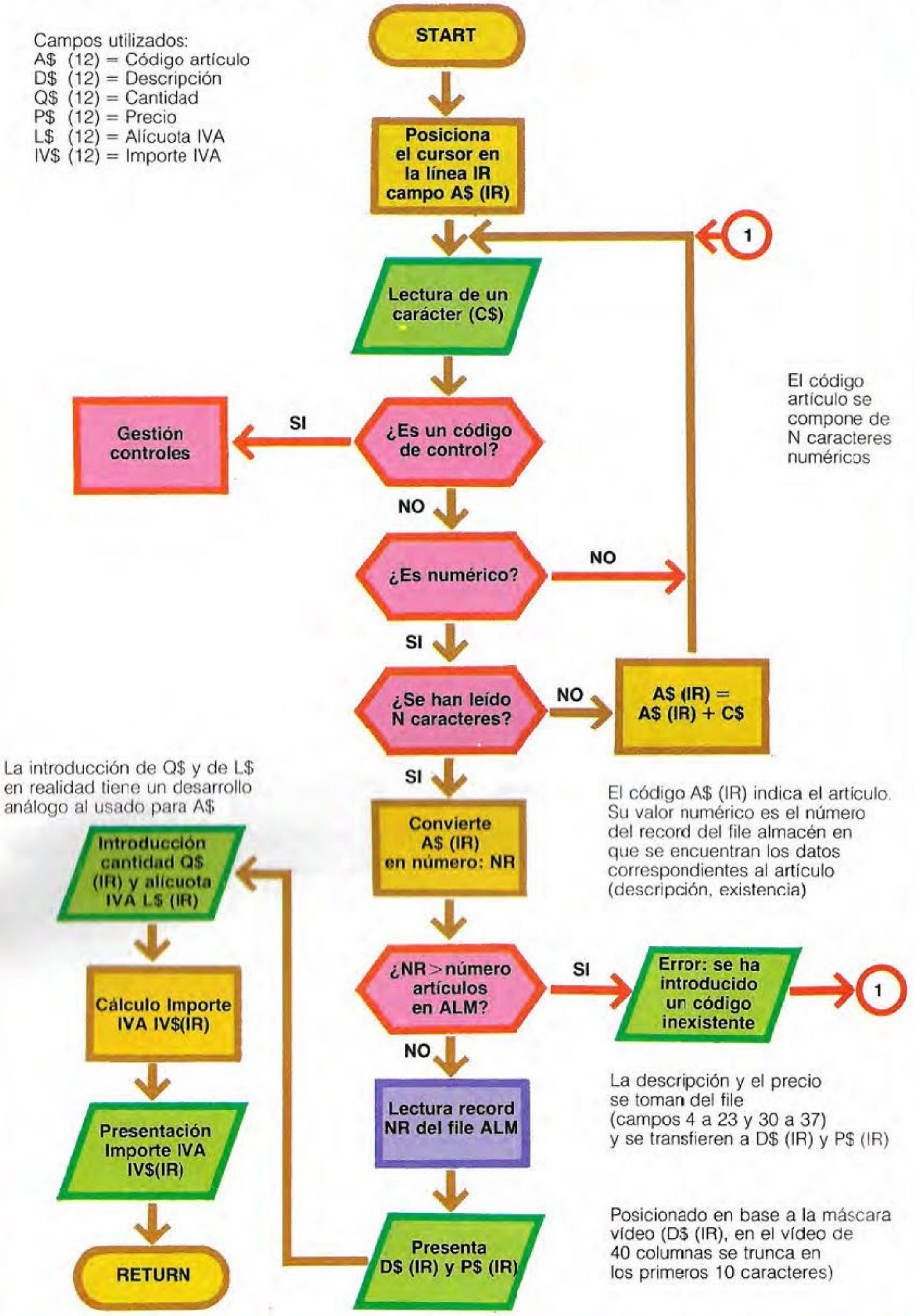
Por tanto, debe preverse una rutina que, según el caso, complete el campo predisponiéndolo para la impresión. Por ejemplo, fijando el formato de presentación en cuatro enteros y dos decimales; en este caso, las salidas deberán ser respectivamente: 125.00, 125.00, 125.60, 125.68, 125.68.
Este resultado puede obtenerse con la instrucción PRINT USING, la cual no está presente en

Sistema de proceso Intertec Computer.



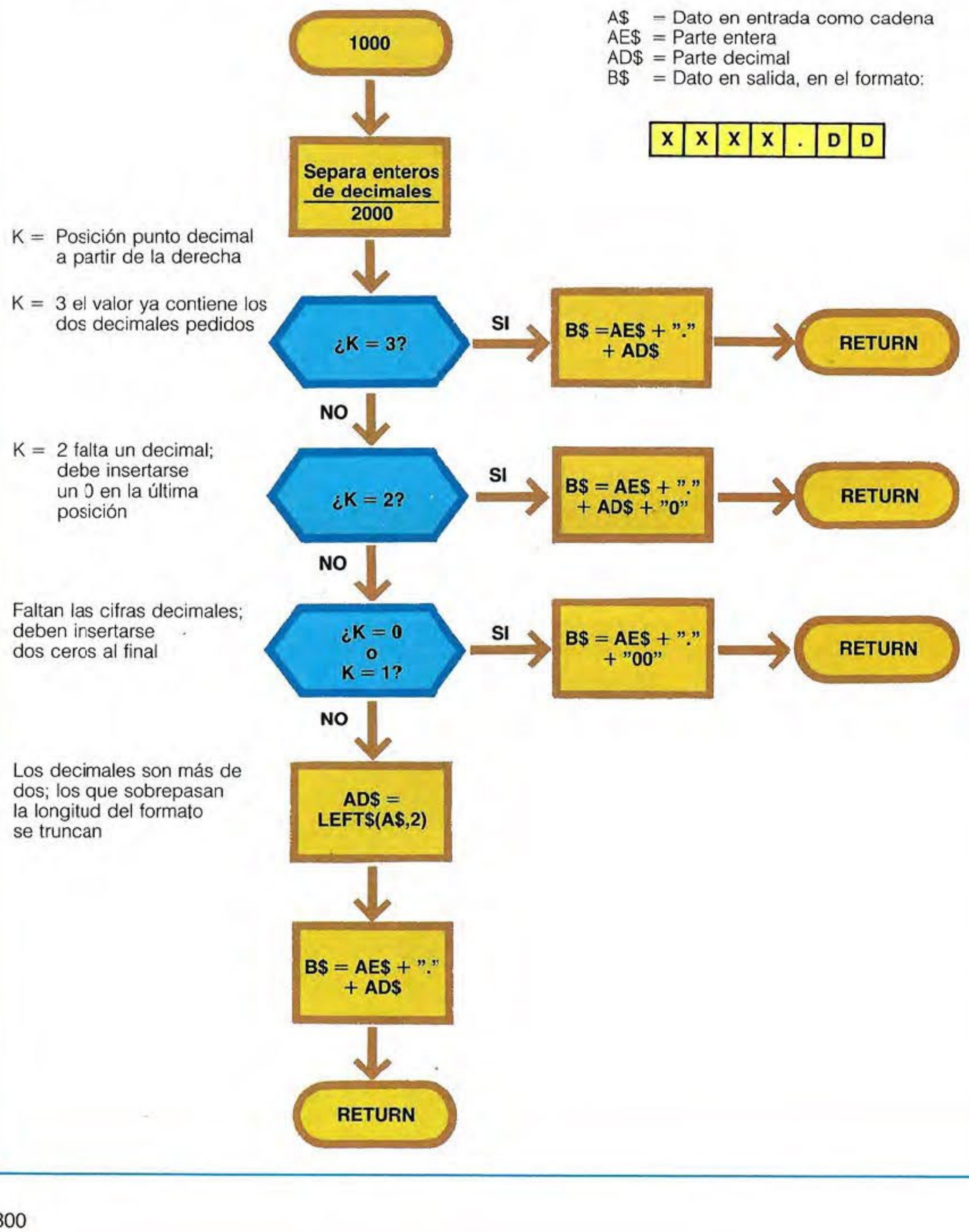
Marka/Photri

DIAGRAMA DE FLUJO PARA LA GESTION DE UNA LINEA EN ENTRADA

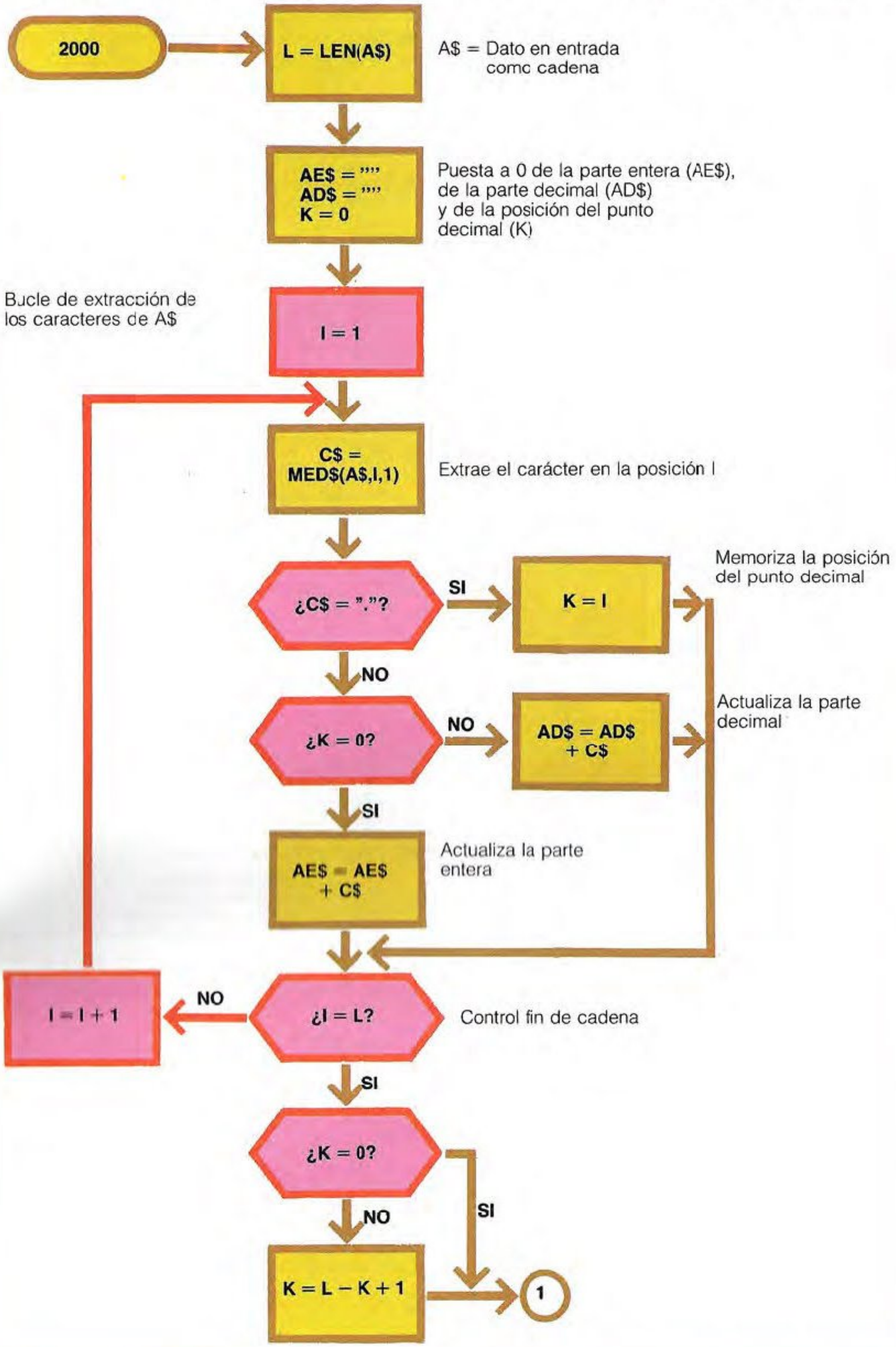


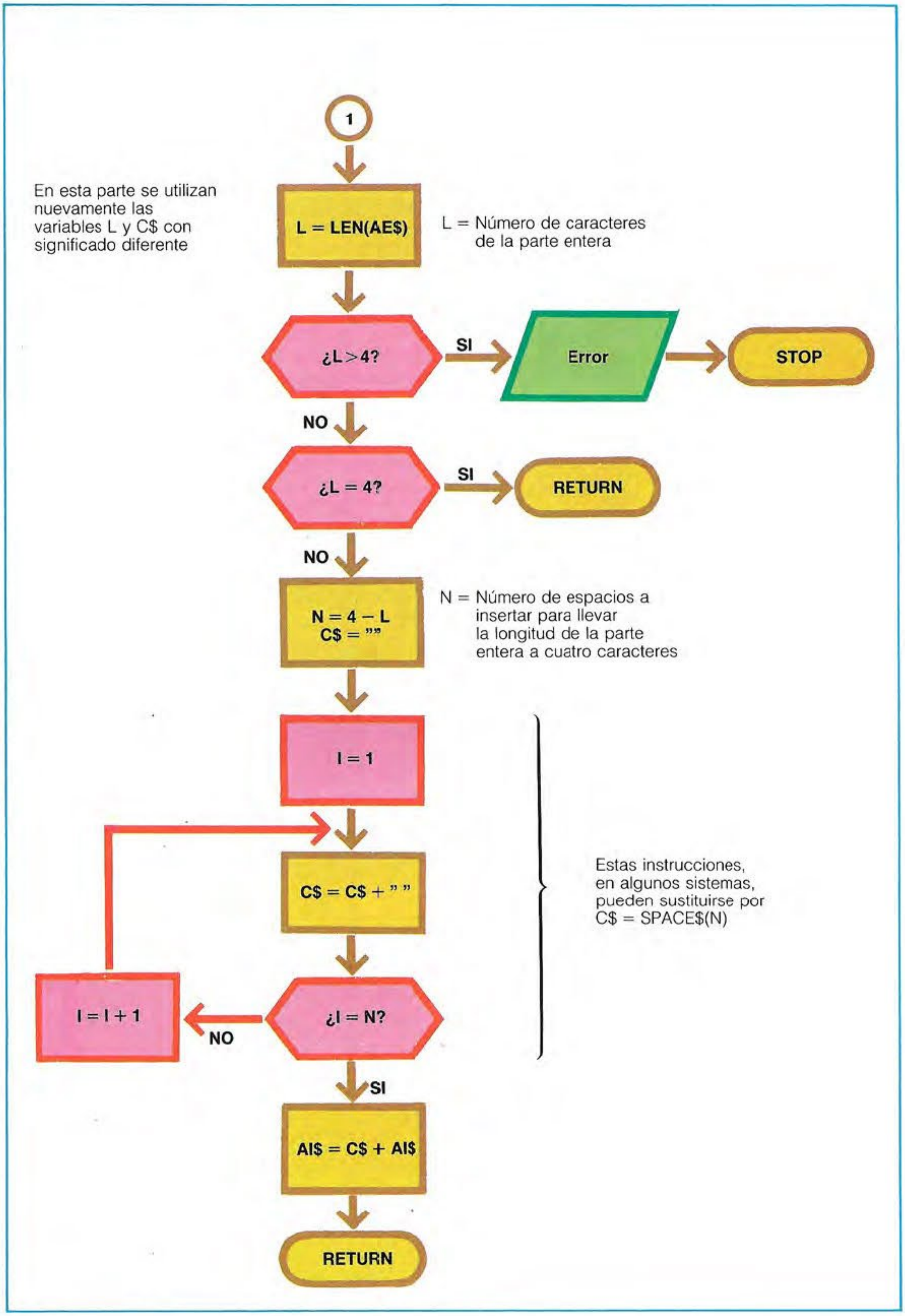
todas las máquinas. En estos casos deberá ser el usuario el que prevea una rutina adecuada. En las figuras de esta página y las dos siguientes se han representado los diagramas de flujo para la preparación del formato de presentación. La subrutina puede adaptarse fácilmente a cada máquina, puesto que las únicas dos funciones intrínsecas utilizadas, LEN y MID$, están presentes en todas las versiones del Basic. Naturalmente, su utilidad está limitada a las versiones del Intérprete que no prevén la opción de impresión con formato (PRINT USING...).

SUBRUTINA PARA LA PREPARACION DE UN FORMATO DE IMPRESION



SUBRUTINA DE SEPARACION DE LA PARTE ENTERA DE LOS DECIMALES

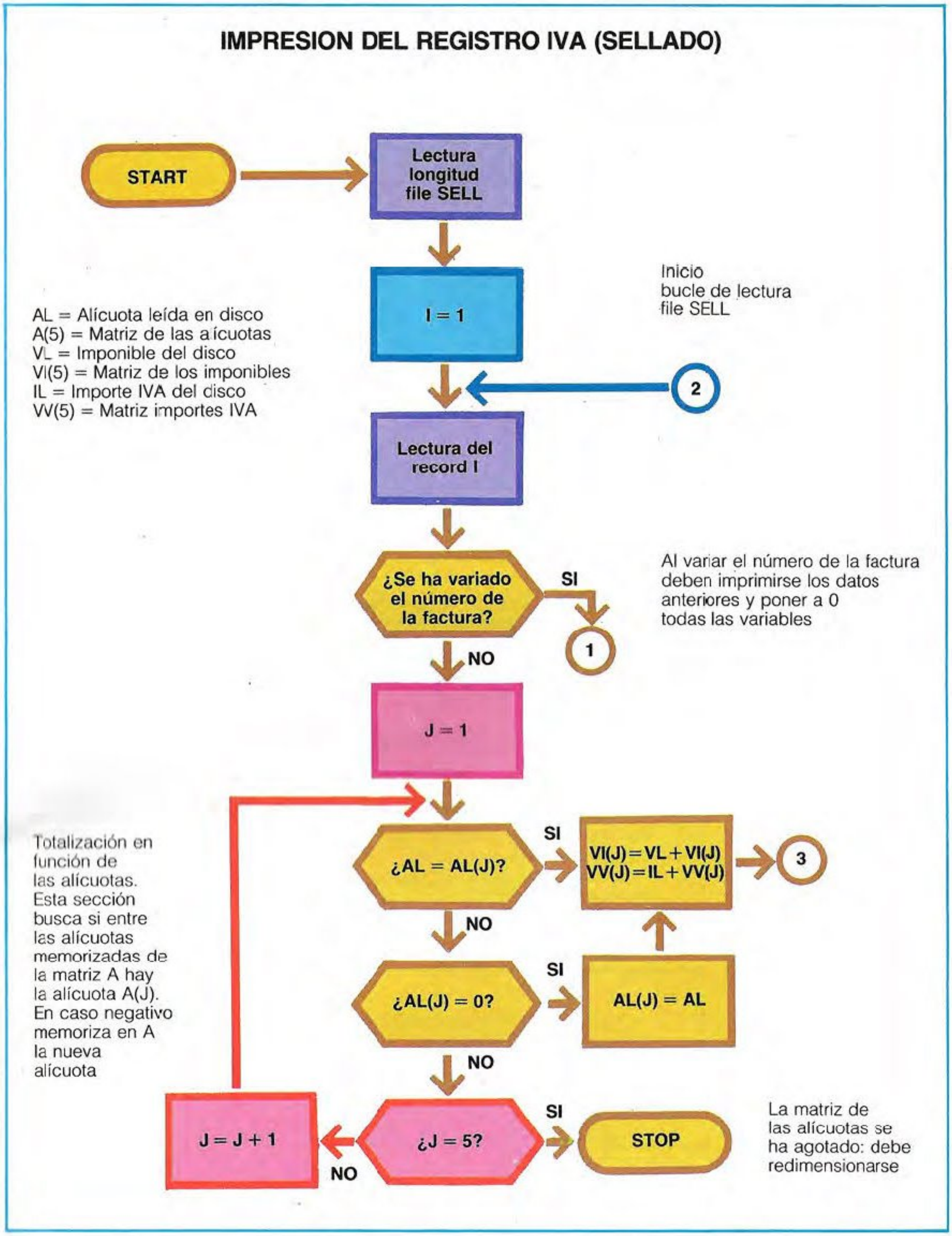


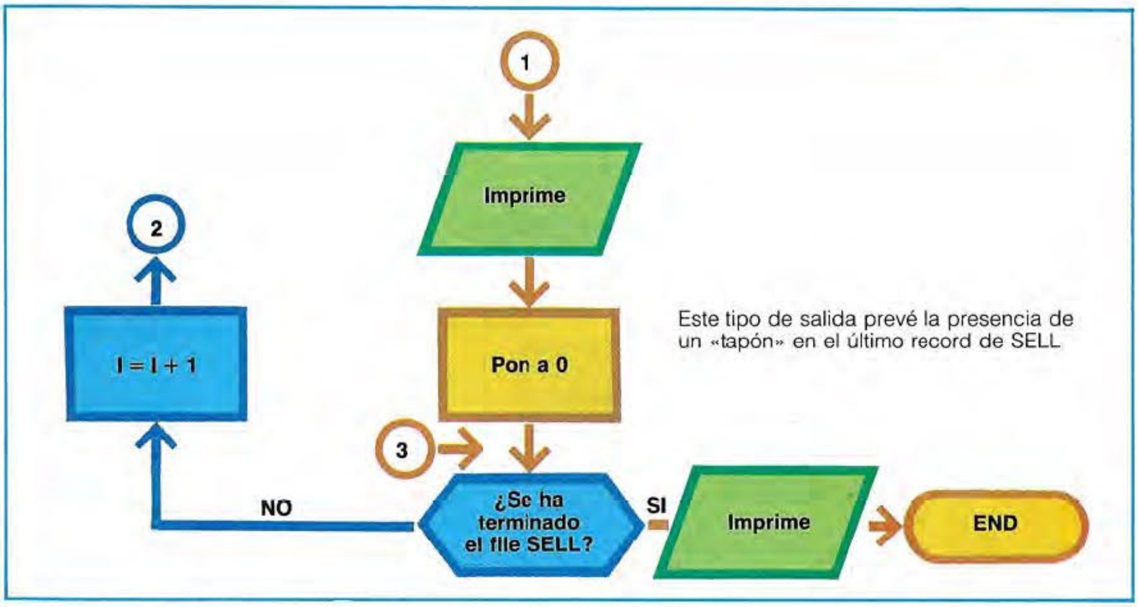


1
En esta parte se utilizan nuevamente las variables L y C$ con significado diferente
L = LEN(AE$)
L = Número de caracteres de la parte entera
¿L > 4?
SI
Error
STOP
NO
¿L = 4?
SI
RETURN
NO
N = 4 − L
C$ = ""
N = Número de espacios a insertar para llevar la longitud de la parte entera a cuatro caracteres
I = 1
C$ = C$ + " "
Estas instrucciones, en algunos sistemas, pueden sustituirse por C$ = SPACE$(N)
I = I + 1
NO
¿I = N?
SI
AI$ = C$ + AI$
RETURN

Impresión del sellado

Con una determinada periodicidad, debe imprimirse un documento (registro sellado, o simplemente sellado) que presente para cada factura el nombre del cliente y los importes subdivididos por partes alícuotas. Durante la emisión de las facturas, los datos se memorizan en el file SELL; por tanto, sólo deben agregársele las partes alícuotas.





ADICION DE LOS IMPORTES EN FUNCION DE LA ALICUOTA IVA

Número Factura	Código Cliente	Fecha	Alícuota	Imponible	Importe IVA
26	03	21 × 12	10	10000	1000

La primera lectura encuentra la matriz A(J) vacía. Por tanto, sitúa la alícuota 10 en la primera posición (J = 1). Al terminar el primer giro (J = 1) se tiene:

A(1) = 10
VI(1) = 10000
VV(1) = 1000

El segundo record tiene la misma alícuota. El programa acumula los datos en la misma posición

A(1) = 10 (la alícuota es un indicador, no debe sumarse)
VI(1) = 35000 (10000+25000)
VV(1) = 3500

26	03	21 × 12	10	25000	2500

El tercer record contiene una alícuota diferente, que por tanto se memoriza en la posición siguiente a la última ocupada (J = 2), obteniendo:

A(2) = 8
VI(2) = 20000
VV(2) = 1600

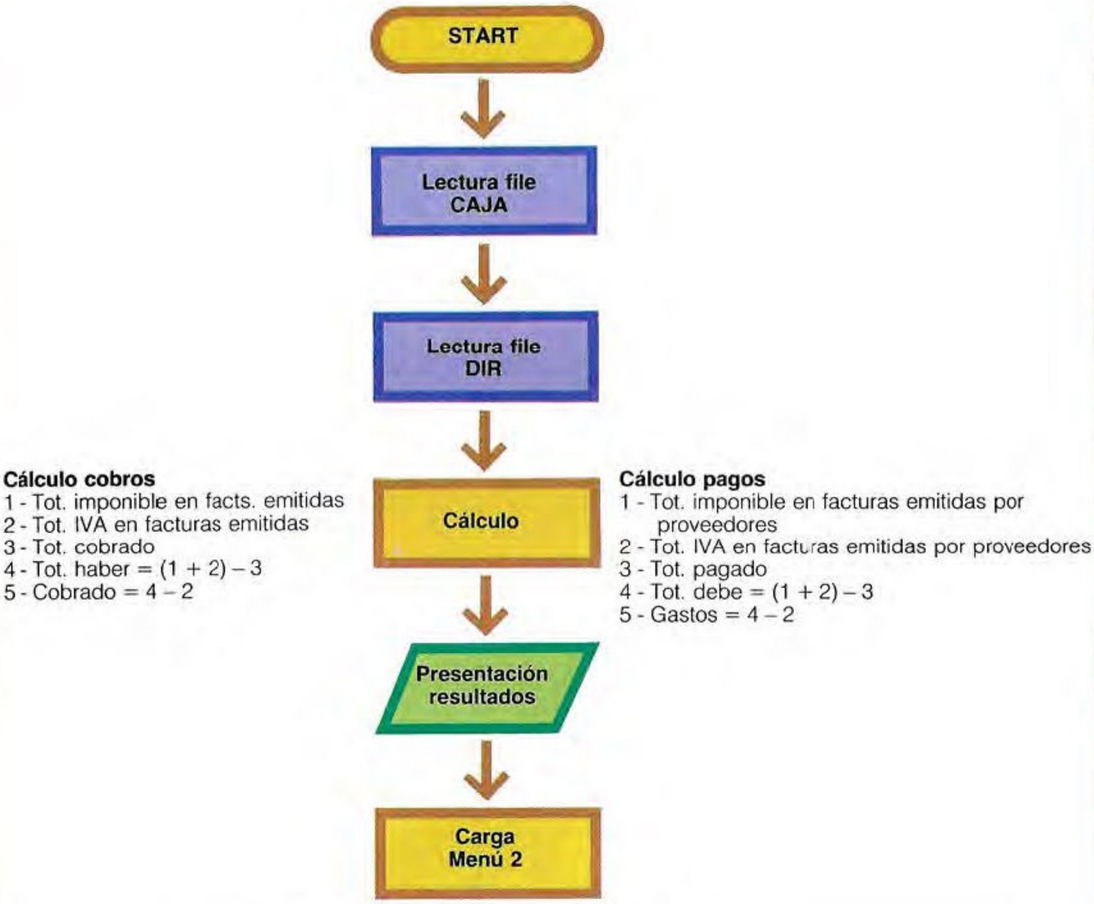
26	03	21 × 12	8	20000	1600

El método, mostrado en las figuras de las págs. 1303 y 1304, utiliza el número de factura como código, cuya ruptura (variación) provoca la impresión de las adiciones correspondientes a aquel número, su puesta a 0 y el inicio de un nuevo ciclo. Además de la impresión del sellado, la rutina también debe utilizarse al final de la compilación de cada factura, puesto que debe presentar, factura por factura, los importes (imponible, IVA) para cada parte alícuota presente. Este mismo tipo de impresión se ha previsto al final del año, para el recopilado anual.

Otras funciones

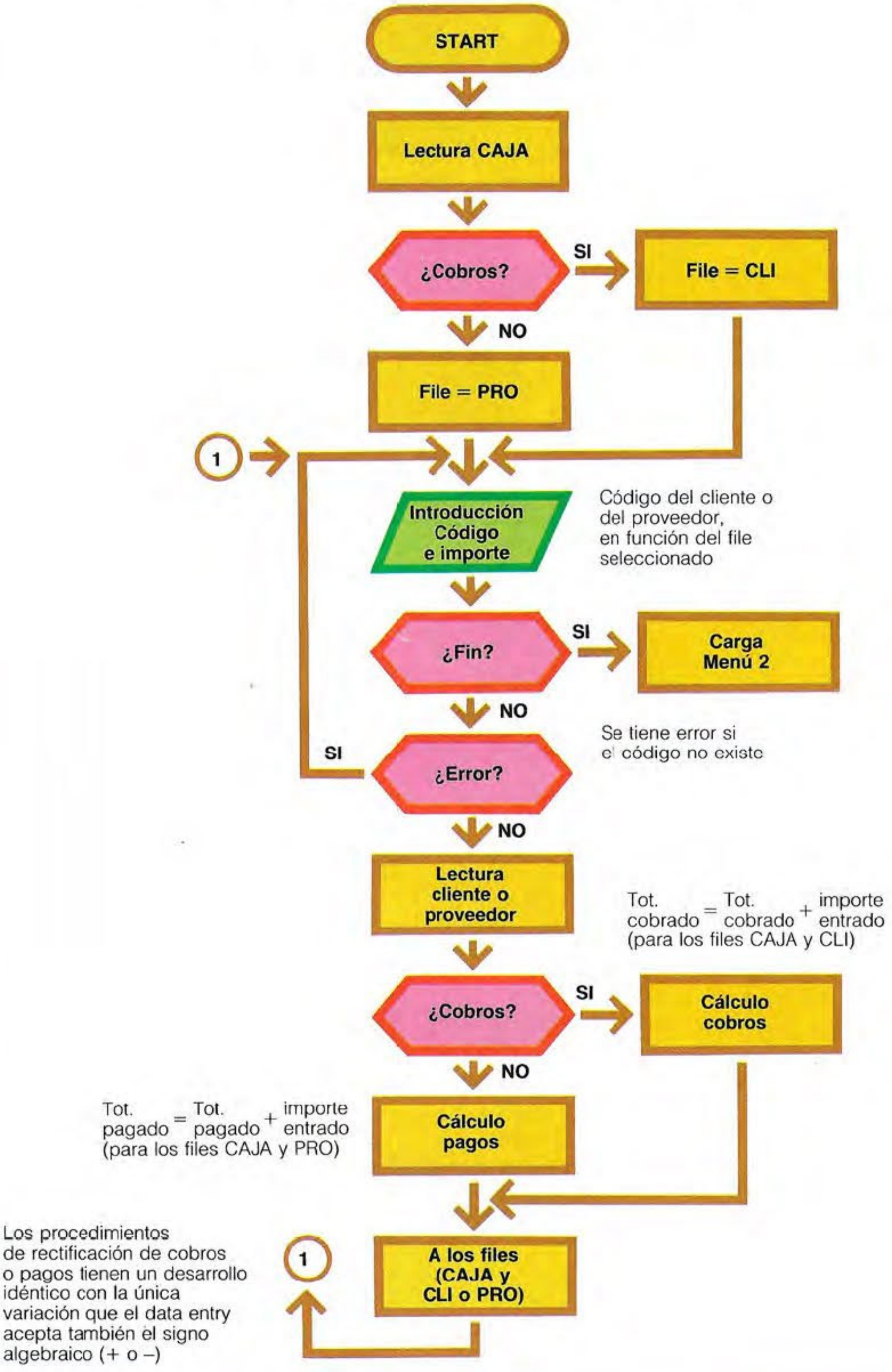
Los otros procedimientos necesarios para completar el programa pueden obtenerse modificando oportunamente los presentados. A título de ejemplo, en las figuras se han representado los diagramas de flujo de dos funciones (Situación de Caja y Cobros/Pagos); como puede verse, son análogos a los anteriores, y su escritura no presenta ninguna dificultad. Además de los módulos o las subrutinas necesarias para esta aplicación particular (de naturaleza dedicada y

DIAGRAMA DE FLUJO DE LA SITUACION DE CAJA



Beneficios – 5 de Cobros – 5 de Pagos
IVA (adeudo/pagar) = 2 de Cobros – 2 de Pagos

DIAGRAMA DE FLUJO COBROS/PAGOS



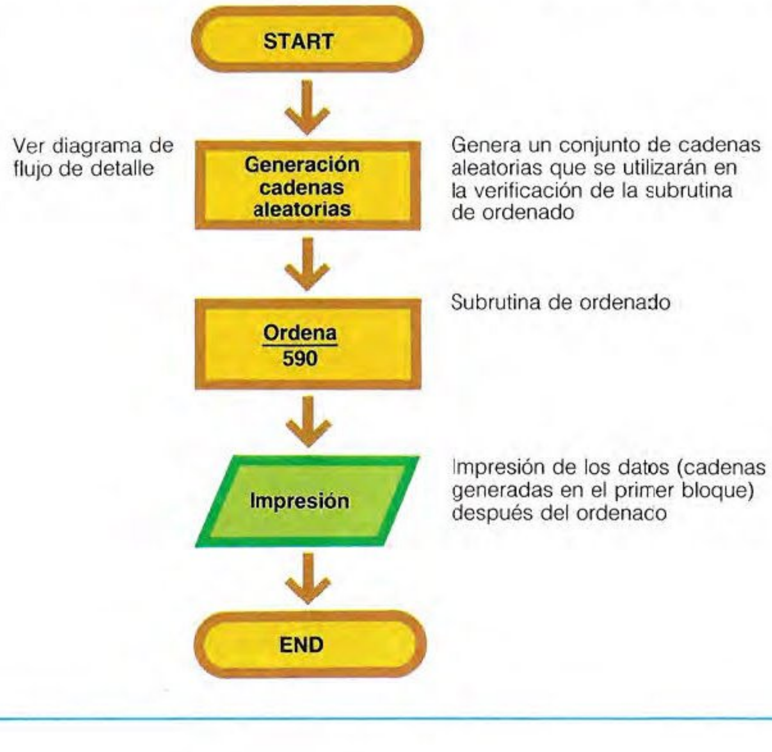
no adaptable a situaciones diferentes) son necesarias algunas rutinas de servicio que realizan funciones de utilidad general, como por ejemplo el Sort.
El ordenado de los datos es una función utilizada frecuentemente en las aplicaciones, y por tanto es útil disponer de un programa que pueda adaptarse a circunstancias diversas.
Abajo se ha representado el diagrama de flujo de principio de un programa de ordenado. El primer bloque (generación de cadenas aleatorias, detallado en la pág. 1309) sirve para generar algunas cadenas a las que se aplica el título de control de la rutina de Sort (subrutina 590). En el ejemplo, las cadenas se obtienen automáticamente utilizando el generador de números aleatorios (RND), como puede verse en el listado de la página siguiente.
Como alternativa puede preverse un bucle de introducción por teclado o de lectura en disco. La subrutina 590, llamada en la línea 260 del main, ejecuta el ordenado y restituye el vector IS de los punteros a los datos en orden creciente. El último bloque imprime los valores ordenados. En la pág. 1310 se ha representado el diagrama de flujo de primer nivel de esta subrutina, deta-



B. Coleman/Marka

Sistema de proceso de configuración adaptada a las aplicaciones gráficas.

DIAGRAMA DE FLUJO DE PRINCIPIO DEL PROGRAMA DE ORDENADO



PROGRAMA DE ORDENADO (MAIN)

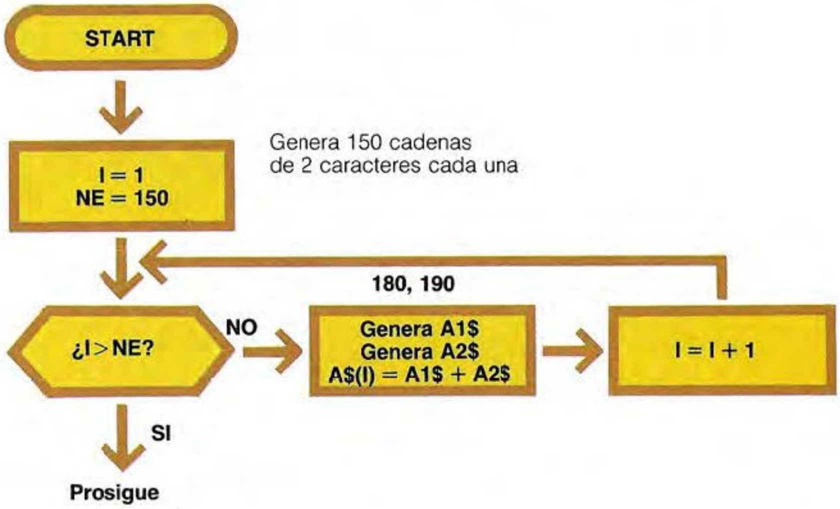
```
10  REM  ********************
20  REM  *                  *
30  REM  *      S O R T     *
40  REM  *                  *
50  REM  ********************
60 :
70  HOME
80  DIM A$(150),IS(160)
90 NE = 150
100 :
110  REM ====================
120  REM *   GENERA  LETRAS   *
130  REM ====================
140 :
150 HTAB 7
160 VTAB 5: PRINT "GENERACION
     LETRAS ALEATORIAS"
170 FOR I = 1 TO NE
180 A1$ = CHR$ (INT (RND (1) *
     26 + 65)):A2$ = CHR$ ( INT
     ( RND (1) * 25 + 65))
190 A$(I) = A1$ + A2$
200  NEXT I
210 :
220  REM ====================
230  REM *    RUTINA SORT     *
240  REM ====================
250 :
260 GOSUB 590
270 :
280 :
290  REM ====================
300  REM *      IMPRESION     *
310  REM ====================
320 :
330  HOME
340 X = 2:Y = :
350 FOR I = 1 TO NE
360 K = IS(I):C$ = A$(K)
370 HTAB X:VTAB Y:PRINT C$
380 IF Y = 23 THEN Y = 1:X = X+
     3 : GOTO 400
390 Y = Y + 2
400 NEXT I
410 VTAB 24: HTAB 38:GET Q$
420 NORMAL
430 HOME : VTAB 24
440 END
450:
```

llada en el diagrama de flujo de segundo nivel de las págs. 1311 a 1313.

La lógica utilizada es la de la búsqueda binaria. El primer paso a realizar es el cálculo de M, puntero utilizado para el control. Inicialmente, M es igual a la mitad del número de los datos ordenados (la primera vez es 1).

Sucesivamente se van tomando uno por uno los

GENERACION DE CADENAS ALEATORIAS



elementos de la matriz a ordenar con un bucle entre 1 y NE (número de datos, instrucciones 640 a 1440 del listado).

Cada elemento se compara con el apuntado por M. Por ejemplo, con I = 25 y M = 6 se compara el elemento A$(25) con el A$(6).

En base al resultado de la comparación se determina si la posición del elemento I en el lector de los punteros se ha encontrado o si debe realizarse una nueva búsqueda. En este último caso se pasa a un nuevo cálculo de M, después de haber establecido si debe desplazarse a la derecha o a la izquierda.

Si la posición se ha encontrado, se memoriza el puntero del elemento en el vector IS, trasladando eventualmente los punteros insertados anteriormente.

Al final, la subrutina de ordenación ha memorizado en IS los punteros a los elementos de A$, según el orden creciente. Para obtener la impresión ordenada debe extraerse de A$ cada elemento según el orden memorizado en IS.

Por ejemplo, suponiendo que IS contenga

IS(1) = 7, IS(2) = 9, IS(3) = 1, IS(4) = 15, ...

el primer elemento a imprimir es A$(7), el segundo A$(9) y así sucesivamente.

El listado representado trabaja en memoria y no prevé la parametrización de los campos a seleccionar.

La primera modificación es muy sencilla de obtener. A las líneas 710, 730, 1130 y 1240 de las variables de cadena IV$ e I1$ se transfiere un elemento de la matriz A$ que se presupone en memoria. Para utilizar datos en el disco es suficiente modificar esas instrucciones sustituyendo la asignación sencilla, por ejemplo IV$ = A$(I), con una lectura en el record apuntado por el índice de A$; así, la instrucción 710 (y análogamente las otras) debe sustituirse por:

— lectura del record I
— transferencia de los datos leídos a IV$

La 730 debe realizar la misma lógica, con la única diferencia en el número de record, que se convierte en J.

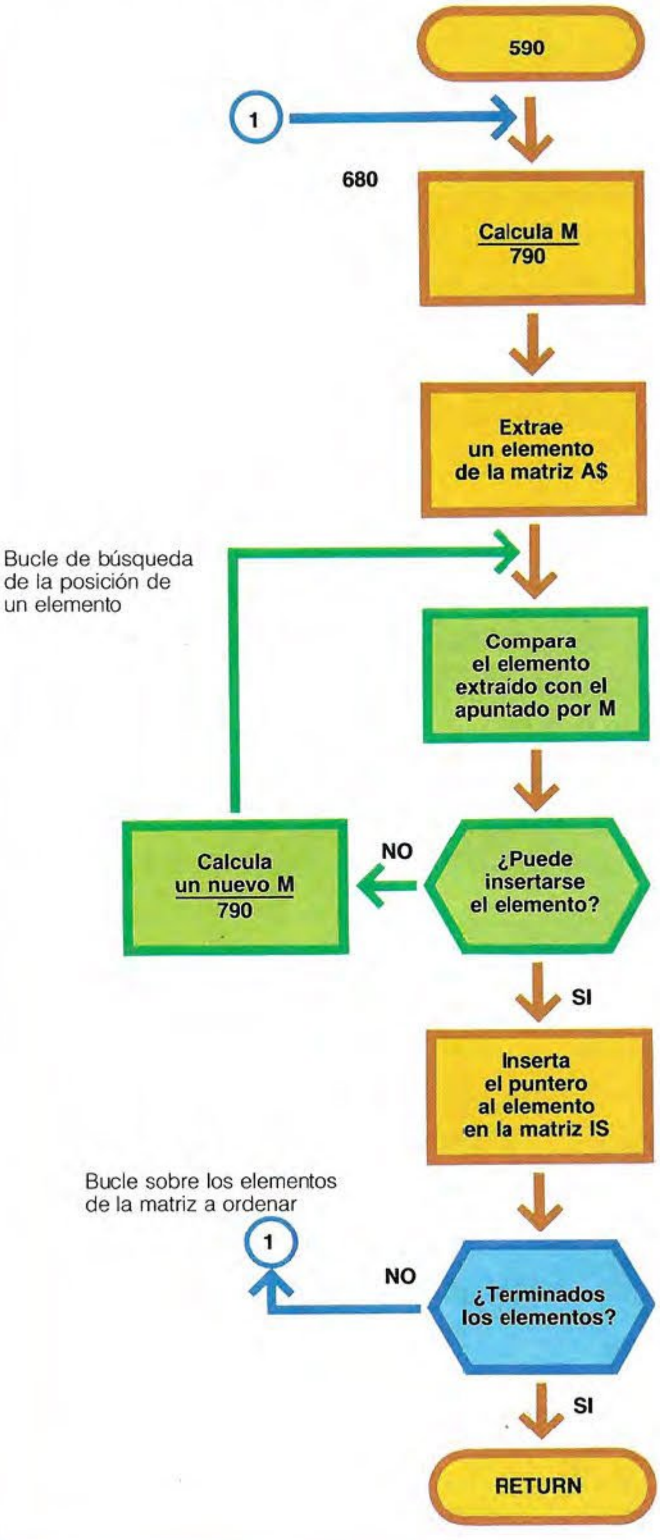
La segunda modificación (parametrización del campo utilizado para el ordenado) puede obtenerse conjuntamente con la primera.

En la lectura de un record, los datos se transfieren en primer lugar a un buffer de apoyo, y seguidamente se toman para la sola posición que interesa y se transfieren a las variables IV$ o I1$.

En la pág. 1315 se ha representado un diagrama de flujo que ilustra este método.

Naturalmente, al utilizar datos en disco debe modificarse también la impresión, puesto que antes de la transferencia a las variables C$ (línea 360) se hace necesaria una fase de lectura en disco.

DIAGRAMA DE FLUJO DE PRINCIPIO DE LA SUBRUTINA 590

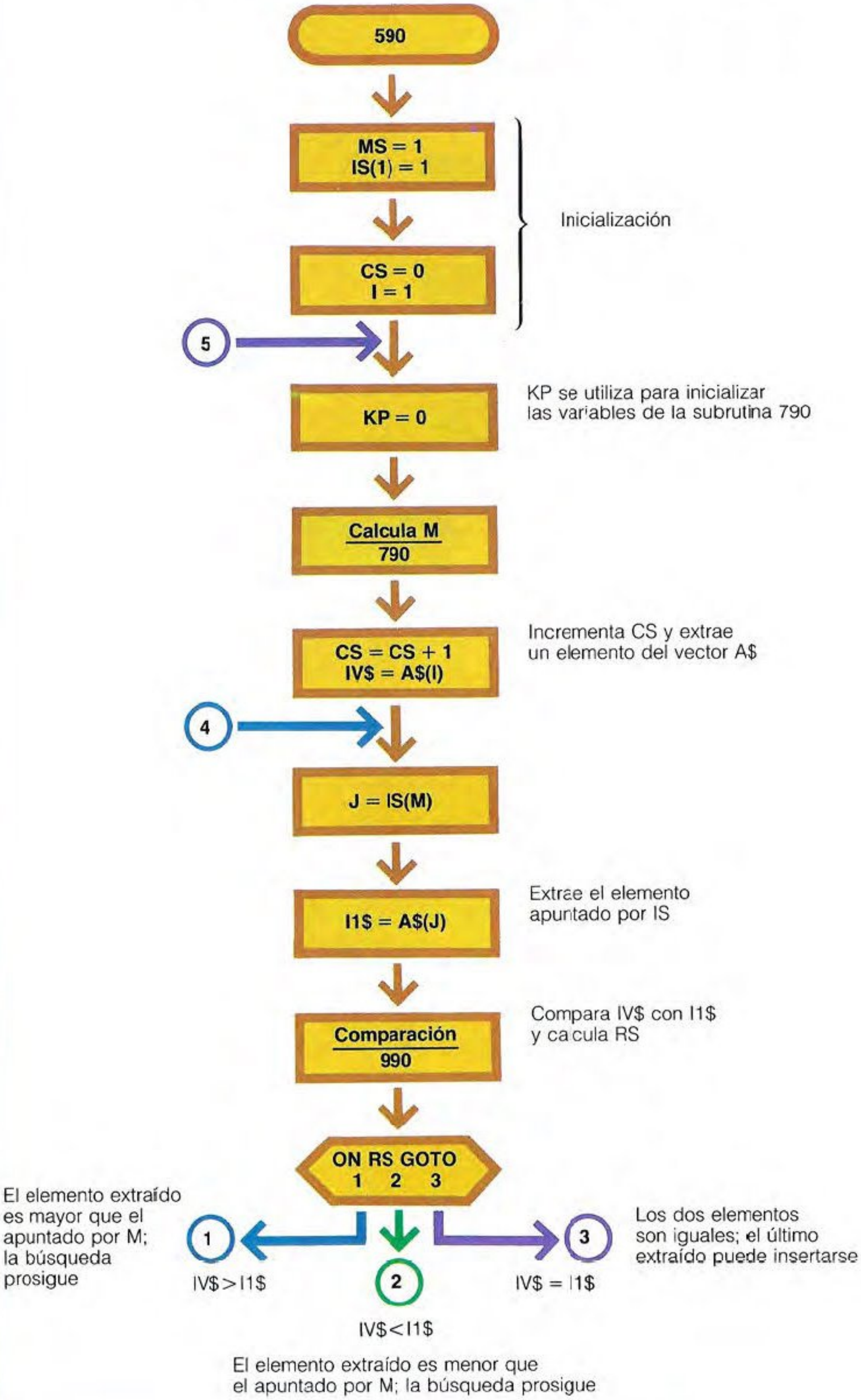


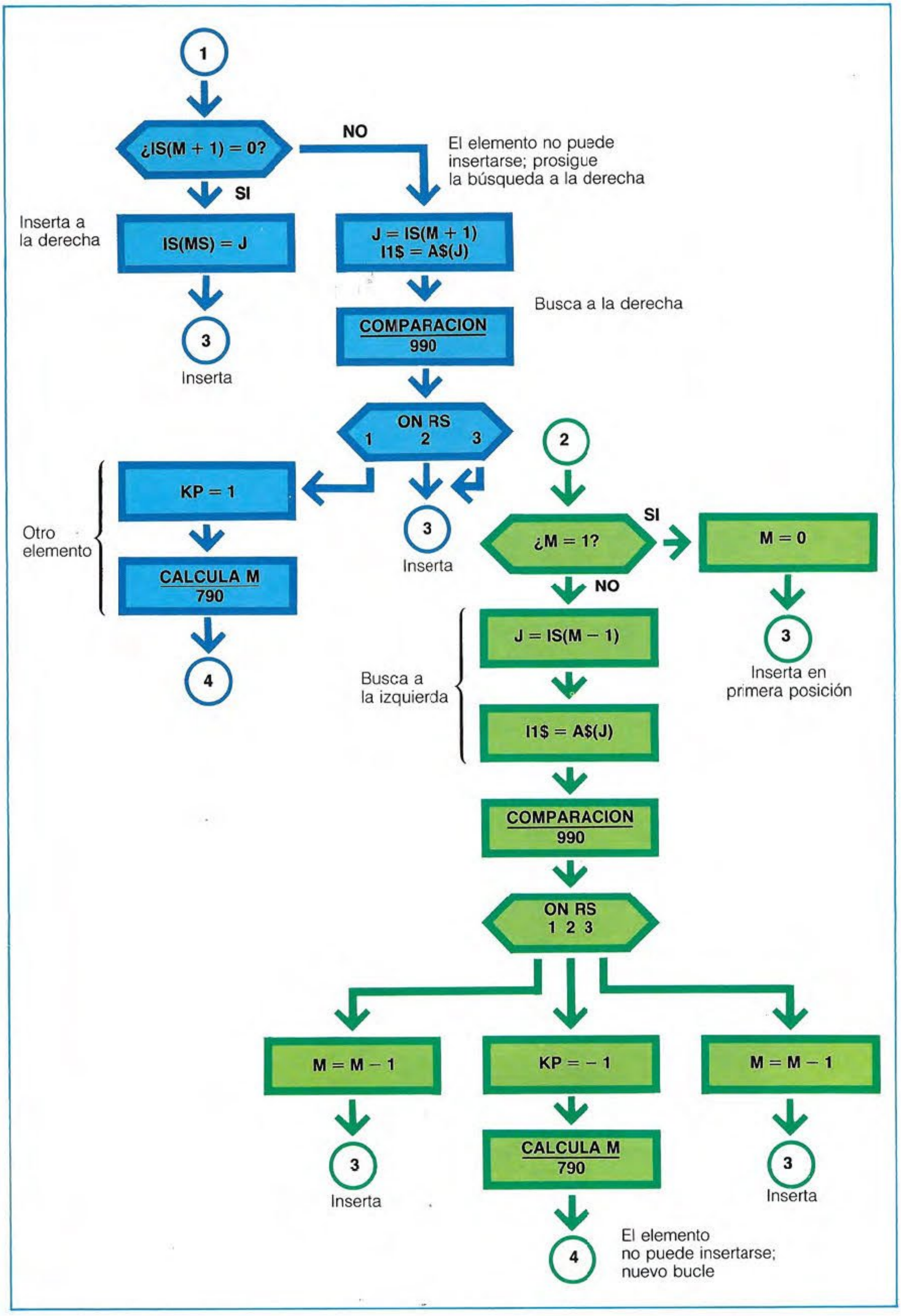
Entrada: Matriz A$ que contiene los elementos a ordenar

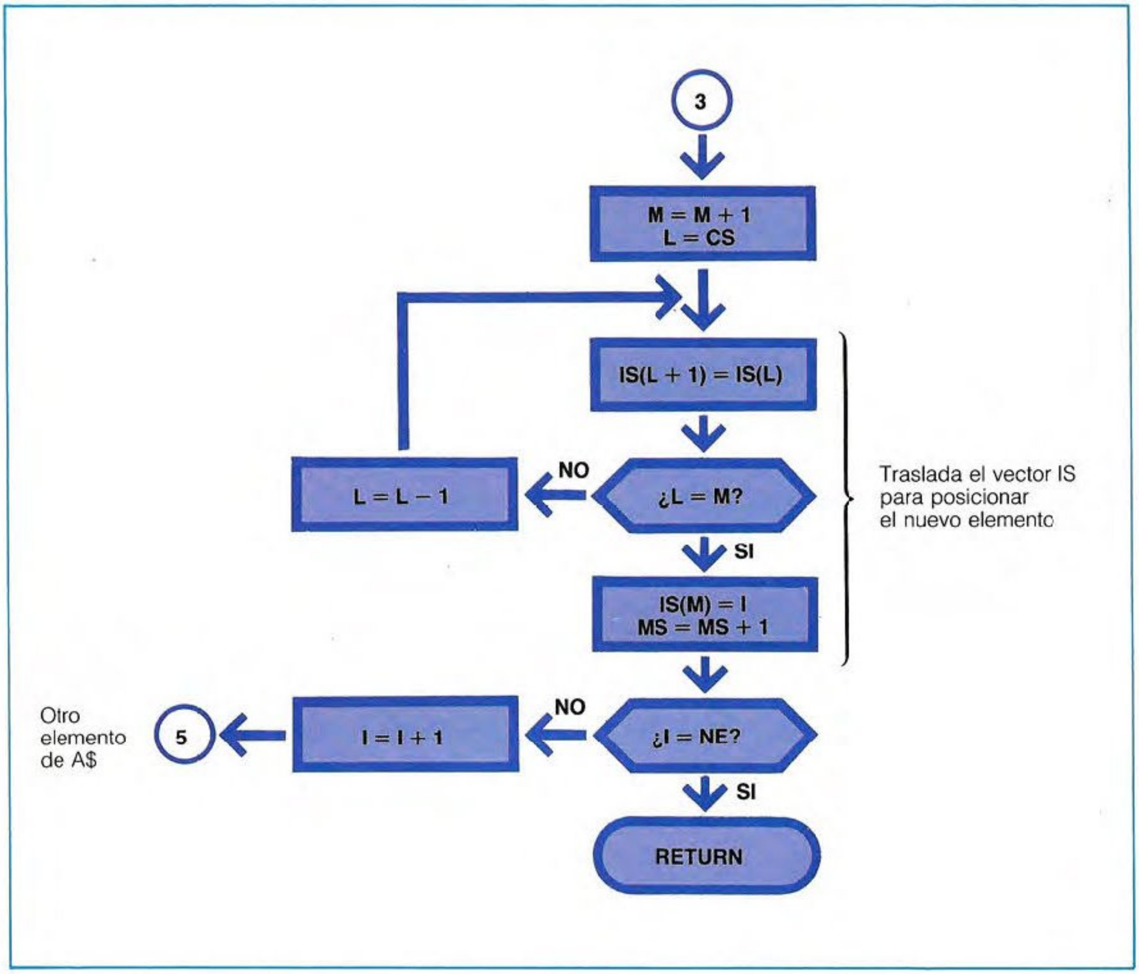
Proceso: Ordenado

Salida: Matriz IS de los punteros a los datos ordenados

DIAGRAMA DE FLUJO DE LA SUBRUTINA 590




1
¿IS(M + 1) = 0?
NO
El elemento no puede insertarse; prosigue la búsqueda a la derecha
SI
Inserta a la derecha
IS(MS) = J
J = IS(M + 1)
I1$ = A$(J)
3
Inserta
Busca a la derecha
COMPARACION
990
ON RS
1 2 3
2
KP = 1
Otro elemento
CALCULA M
790
4
3
Inserta
¿M = 1?
SI
M = 0
NO
3
Inserta en primera posición
Busca a la izquierda
J = IS(M − 1)
I1$ = A$(J)
COMPARACION
990
ON RS
1 2 3
M = M − 1
KP = − 1
M = M − 1
3
Inserta
CALCULA M
790
3
Inserta
4
El elemento no puede insertarse; nuevo bucle



PROGRAMA DE ORDENADO (SUBRUTINAS)

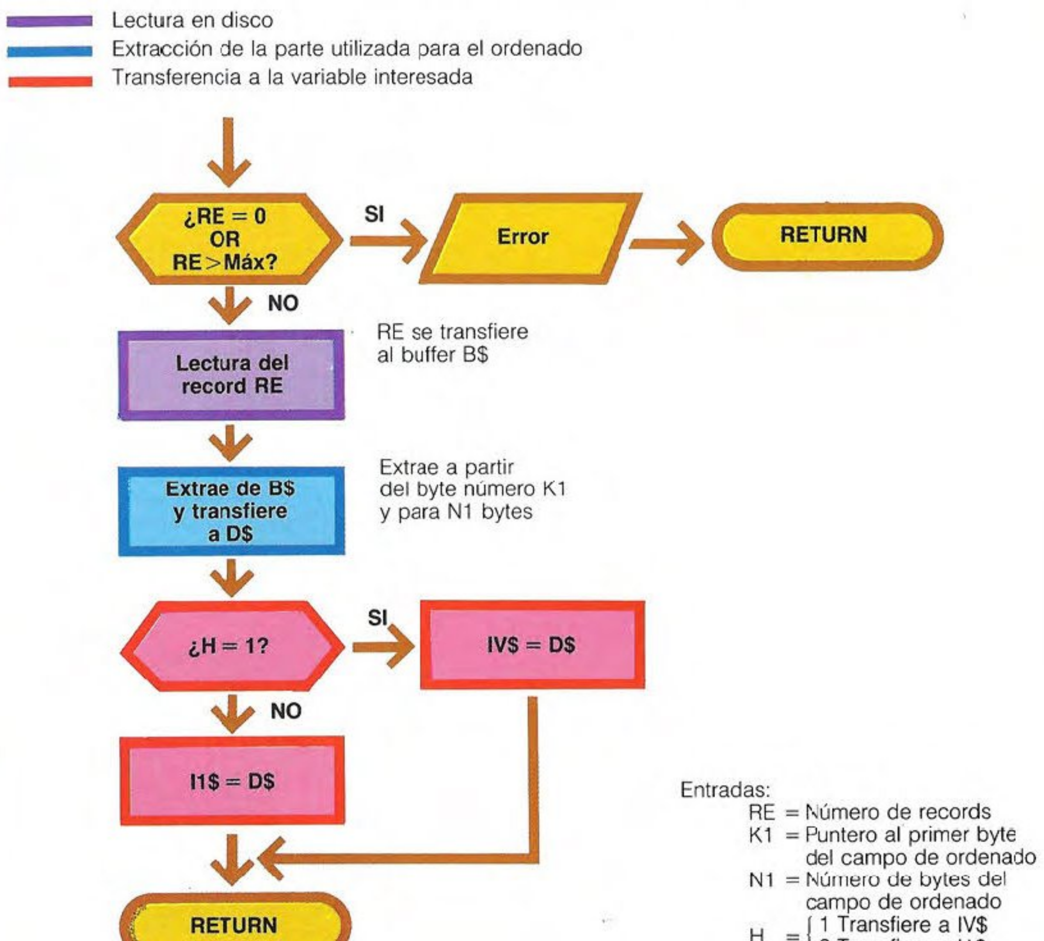
```
460  REM  ========================
470  REM  *          SORT          *
480  REM  ========================
490 :
500  REM  ============================
510  REM  *IS(.)=VECTOR ORDENADO*
520  REM  *MS=MAX PUNTERO IS()*
530  REM  *CS=N. ELEMENTOS ORDENADOS.*
540  REM  *A$()=VECTOR A ORDENAR*
550  REM  *IV$=ELEM. A ORDENAR*
560  REM  *I1$=ELEM. DE COMPARACION*
570  REM  *M=PUNTERO DE BUSQUEDA*
580  REM  ============================
590 MS = 1:IS(1) = 1:CS = 0
600  HOME : VTAB 10: INVERSE : PRINT
     "ORDENO EL ELEMENTO N': "
```

```
610  REM  ======================
620  REM   *  CICLO ORDENADO  *
630  REM  ======================
640  FOR I = 1 TO NE
650  HTAB 24: VTAB 10: PRINT I
660 KP = 0
670 :
680  GOSUB 790: REM  * M *
690 :
700 CS = CS + 1
710 IV$ = A$(I)
720  REM
730 J = IS(M):I1$ = A$(J)
740 :
750  GOSUB 990: REM  * CONF. *
760 :
770  ON RS GOTO 1080.1190.1320
780 :
790  REM ======================
800  REM *   CALCULO DE (M)    *
810  REM ======================
820 :
830 :IF KP <   > 0 THEN 870
840 MP = 1:PP = MS:PS = 0:M = INT
     (MS / 2)
850  IF M = 0 THEN M = 1
860  RETURN
870  IF KP = 1 THEN 900
880  IF KP = - 1 THEN 950
890 :
900 MP = M: IF PP = 0 THEN PP = M
     S
910 M2 = INT ((PP - M) / 2) : IF
     M2 = 0 THEN M2 = 1
920 M = M + M2
930 :
940  RETURN
950 PP = M:M2 = INT ((M - MP) /
     2): IF M2 = 0 THEN M2 = 1
960 M = M - M2
970  RETURN
980 :
990  REM ====================
1000  REM  *COMPARA ELEMENTOS*
1010  REM  ====================
1020 :
1030 RS = 3
1040  IF IV$ > I1$ THEN RS = 1
1050  IF IV$ < I1$ THEN RS = 2
1060  RETURN
1070 :
1080  REM  =====================
1090  REM  *      IV$ > I1$      *
1100  REM  =====================
1110 :
1120  IF IS (M + 1) = 0 THEN IS(MS
     ) = J:GOTO 1320
1130 J = IS(M + 1):I1$ = A$(J)
1140  GOSUB 990
1150  ON RS GOTO 1160.1320.1320
1160 KP = 1
1170  GOSUB 790: GOTO 720
1180 :
1190  REM ======================
1200  REM *     IV$ < I1$       *
1210  REM ======================
1220 :
1230  IF M = 1 THEN M = 0: GOTO 1
     320
1240 J = IS (M - 1):I1$ = A$(J)
1250   GOSUB 990
```

```
1260  ON RS GOTO 1270,1280,1270
1270 M = M - 1: GOTO 1320
1280 KP = - 1
1290  GOSUB 790
1300  GOTO 720
1310 :
1320  REM ====================
1330  REM *     IV$ = I1$     *
1340  REM ====================
1350 :
1360 M = M + 1
1370 :
1380  REM    * CONTRUCCION DE IS(,)*

1390 :
1400  FOR L = CS TO M STEP - 1
1410 IS(L + 1) = IS(L)
1420  NEXT L
1430 IS(M) = I:MS = MS + 1
1440  NEXT I
1450  RETURN
```

GENERALIZACION DE LA SUBRUTINA DE SORT



La transmisión de las informaciones

En el ámbito de la informática, el concepto de transmisión de datos se entiende como el envío de informaciones codificadas de un punto a otro mediante sistemas de comunicación de tipo electrónico o electromecánico. La distancia que separa los dos puntos de intercambio no está sujeta a restricciones, a no ser en la medida en que tal distancia condiciona la elección del sistema de transmisión. En todos los sistemas de comunicación tradicionales (teléfono, radio, televisión, etc.), la información (voz, imagen, texto escrito) se codifica en la fuente en señales electromagnéticas y se transmite a través de un medio adecuado (el cable de una línea telefónica, el éter). En la recepción, las señales vuelven a convertirse de manera que restituyan la forma inicial a las informaciones.

Ahora se enfocará la atención en las comunicaciones de las informaciones en el caso del ordenador. La primera transferencia es intrínseca al diálogo hombre-máquina. El ordenador trabaja utilizando secuencias de cifras binarias de 0 y 1; por tanto, para comunicarse con el hombre debe traducir cada secuencia de bits en caracteres alfabéticos o numéricos. La traducción se realiza utilizando tablas de correspondencia biunívoca entre caracteres (código hombre) y secuencias de bits (código máquina). Las formas de codificación más difundidas se denominan con las siglas ASCII y EBCDIC.

Las dos codificaciones se diferencian en las combinaciones de bits asociadas a un mismo carácter; las dos incluyen, además de las letras y los números, un cierto número de símbolos especiales, utilizados en las comunicaciones de los datos, de lo que se hablará a continuación. Al lado se representa la codificación ASCII de los caracteres especiales para las comunica-

Sistema de proceso portátil NEC conectado a los periféricos.



R. Villa/Marka

CODIFICACION ASCII DE LOS CARACTERES ESPECIALES UTILIZADOS EN LA TELETRANSMISION

Carácter	Significado	Código ASCII bit n.° 8* 7 6 5 4 3 2 1	Valor decimal	Valor octal	Valor hexadecimal
NUL	Null	0 0 0 0 0 0 0	0	0	0
SOH	Start of Heading	0 0 0 0 0 0 1	1	1	1
STX	Start of Text	0 0 0 0 0 1 0	2	2	2
ETX	End of Text	0 0 0 0 0 1 1	3	3	3
EOT	End of Transmission	0 0 0 0 1 0 0	4	4	4
ENQ	Enquiry	0 0 0 0 1 0 1	5	5	5
ACK	Acknowledgement	0 0 0 0 1 1 0	6	6	6
BEL	Bell	0 0 0 0 1 1 1	7	7	7
BS	Backspace	0 0 0 1 0 0 0	8	10	8
HT	Horizontal Tabulation	0 0 0 1 0 0 1	9	11	9
NL	New Line	0 0 0 1 0 1 0	10	12	A
VT	Vertical Tabulation	0 0 0 1 0 1 1	11	13	B
FF	Form Feed	0 0 0 1 1 0 0	12	14	C
RT	Return	0 0 0 1 1 0 1	13	15	D
SO	Shift Out	0 0 0 1 1 1 0	14	16	E
SI	Shift In	0 0 0 1 1 1 1	15	17	F
DLE	Data Line Escape	0 0 1 0 0 0 0	16	20	10
DC1	Device Control 1	0 0 1 0 0 0 1	17	21	11
DC2	Device Control 2	0 0 1 0 0 1 0	18	22	12
DC3	Device Control 3	0 0 1 0 0 1 1	19	23	13
DC4	Device Control 4	0 0 1 0 1 0 0	20	24	14
NAK	Negative Acknowledgement	0 0 1 0 1 0 1	21	25	15
SYN	Synchronous Idle	0 0 1 0 1 1 0	22	26	16
ETB	End of Transmission Block	0 0 1 0 1 1 1	23	27	17
CAN	Cancel	0 0 1 1 0 0 0	24	30	18
EM	End of Medium	0 0 1 1 0 0 1	25	31	19
SUB	Substitute	0 0 1 1 0 1 0	26	32	1A
ESC	Escape	0 0 1 1 0 1 1	27	33	1B
FS	File Separator	0 0 1 1 1 0 0	28	34	1C
GS	Group Separator	0 0 1 1 1 0 1	29	35	1D
RS	Record Separator	0 0 1 1 1 1 0	30	36	1E
US	Unit Separator	0 0 1 1 1 1 1	31	37	1F

* El bit n.° 8 es el bit de paridad.

ciones de los datos, y en la página siguiente se representa la codificación EBCDIC.

Modalidades y sistemas de comunicación

Ya se ha dicho que los códigos ASCII y EBCDIC emplean ocho bits para representar un carácter; generalmente siete bits contienen la información propiamente dicha, mientras que el bit más significativo es el bit de paridad.

Para transmitir un carácter entre dos aparatos, por ejemplo un ordenador y un periférico, es necesario conectarlos entre sí con una línea de comunicación; según cómo se realiza la línea son posibles dos tipos diferentes de transmisión:

- **conexión en paralelo**
  La línea está constituida por ocho pistas (hilos) por las que viajan simultáneamente los ocho bits que componen la información.
- **conexión en serie**
  La línea está constituida por una sola pista (hilo) en la que los bits que componen la información viajan uno después de otro.

Las dos posibilidades se han esquematizado en la figura de la pág. 1319.

Como puede intuirse, con la conexión de tipo paralelo pueden obtenerse velocidades de transmisión más elevadas; por tanto, este tipo de conexión es típico de los periféricos rápidos (unidad de disco, unidad de cinta, impresoras rápidas, etc.), y en cualquier caso no alejados del ordenador (distancias de algunos metros), a causa del costo de la conexión. En lo que respecta a la conexión en serie, si por una parte no permite la consecución de velocidades de transmisión elevadas, por otra permite conectar periféricos a distancias importantes haciendo uso de las líneas telefónicas.

Es sobre este aspecto que se enfocará la expo-

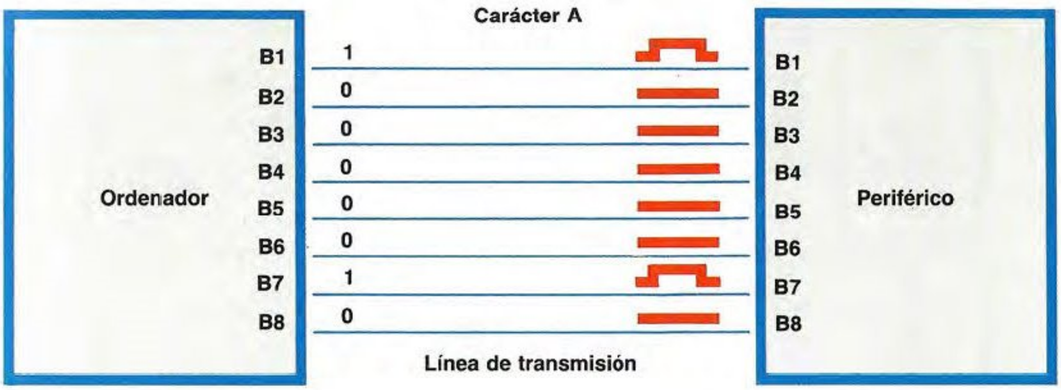
CODIFICACION EBCDIC DE LOS CARACTERES ESPECIALES UTILIZADOS EN LA TELETRANSMISION

Carácter	Código EBCDIC bit n.º 8* 7 6 5 4 3 2 1	Valor decimal	Valor octal	Valor hexadecimal
NUL	0 0 0 0 0 0 0	0	0	0
SOH	0 0 0 0 0 0 1	1	1	1
STX	0 0 0 0 0 1 0	2	2	2
ETX	0 0 0 0 0 1 1	3	3	3
PF	0 0 0 0 1 0 0	4	4	4
HT	0 0 0 0 1 0 1	5	5	5
LC	0 0 0 0 1 1 0	6	6	6
DEL	0 0 0 0 1 1 1	7	7	7
	0 0 0 1 0 0 0	8	10	8
RLF	0 0 0 1 0 0 1	9	11	9
SMM	0 0 0 1 0 1 0	10	12	A
VT	0 0 0 1 0 1 1	11	13	B
FF	0 0 0 1 1 0 0	12	14	C
CR	0 0 0 1 1 0 1	13	15	D
SO	0 0 0 1 1 1 0	14	16	E
SI	0 0 0 1 1 1 1	15	17	F
DLE	0 0 1 0 0 0 0	16	20	10
DC1	0 0 1 0 0 0 1	17	21	11
DC2	0 0 1 0 0 1 0	18	22	12
DC3	0 0 1 0 0 1 1	19	23	13
RES	0 0 1 0 1 0 0	20	24	14
NL	0 0 1 0 1 0 1	21	25	15
BS	0 0 1 0 1 1 0	22	26	16
IDL	0 0 1 0 1 1 1	23	27	17
CAN	0 0 1 1 0 0 0	24	30	18
EM	0 0 1 1 0 0 1	25	31	19
CC	0 0 1 1 0 1 0	26	32	1A
CU1	0 0 1 1 0 1 1	27	33	1B
IFS	0 0 1 1 1 0 0	28	34	1C
IGS	0 0 1 1 1 0 1	29	35	1D
IRS	0 0 1 1 1 1 0	30	36	1E
IUS	0 0 1 1 1 1 1	31	37	1F
DS	0 1 0 0 0 0 0	32	40	20
SOS	0 1 0 0 0 0 1	33	41	21
FS	0 1 0 0 0 1 0	34	42	22
	0 1 0 0 0 1 1	35	43	23
BYP	0 1 0 0 1 0 0	36	44	24
LF	0 1 0 0 1 0 1	37	45	25
EOB/ETB	0 1 0 0 1 1 0	38	46	26
PRE/ESC	0 1 0 0 1 1 1	39	47	27
	0 1 0 1 0 0 0	40	50	28
	0 1 0 1 0 0 1	41	51	29
SM	0 1 0 1 0 1 0	42	52	2A
CU2	0 1 0 1 0 1 1	43	53	2B
	0 1 0 1 1 0 0	44	54	2C
ENQ	0 1 0 1 1 0 1	45	55	2D
ACK	0 1 0 1 1 1 0	46	56	2E
BEL	0 1 0 1 1 1 1	47	57	2F
	0 1 1 0 0 0 0	48	60	30
	0 1 1 0 0 0 1	49	61	31
SYN	0 1 1 0 0 1 0	50	62	32
	0 1 1 0 0 1 1	51	63	33
PN	0 1 1 0 1 0 0	52	64	34
RS	0 1 1 0 1 0 1	53	65	35
UC	0 1 1 0 1 1 0	54	66	36
EOT	0 1 1 0 1 1 1	55	67	37
	0 1 1 1 0 0 0	56	70	38
	0 1 1 1 0 0 1	57	71	39
	0 1 1 1 0 1 0	58	72	3A
CU3	0 1 1 1 0 1 1	59	73	3B
DC4	0 1 1 1 1 0 0	60	74	3C
NAK	0 1 1 1 1 0 1	61	75	3D
	0 1 1 1 1 1 0	62	76	3E
SUB	0 1 1 1 1 1 1	63	77	3F

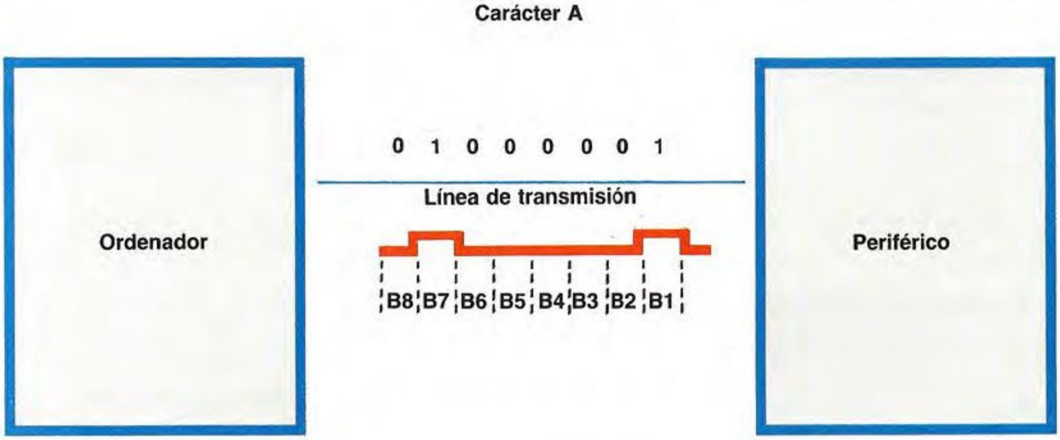
* El bit n.º 8 es el bit de paridad.

TIPOS DE CONEXION

Conexión en paralelo



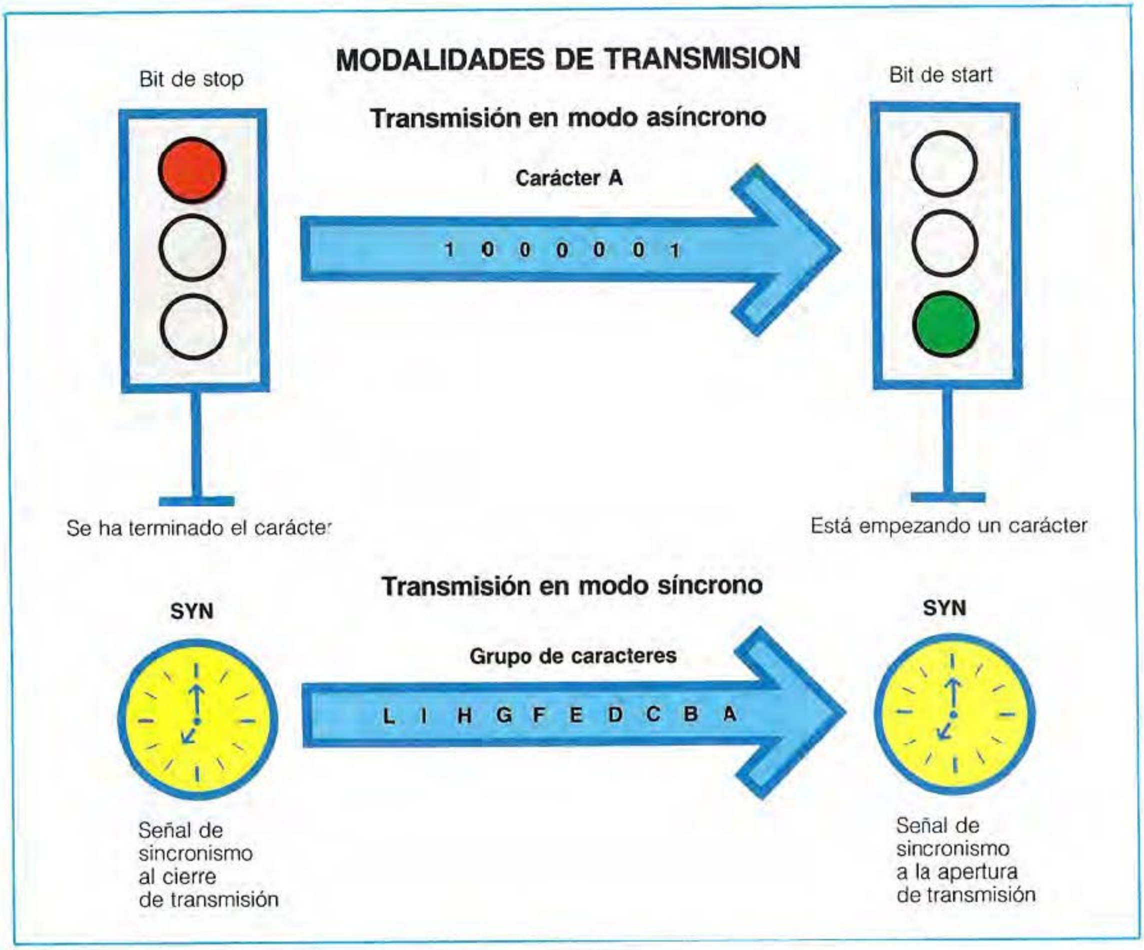
Línea de transmisión



sición de los párrafos siguientes, a causa de la notable importancia práctica que ha alcanzado la transmisión de datos vía cable telefónico.
Ahora examinaremos una conexión de tipo serie y fijaremos otro concepto importante: la modalidad de transmisión. La transmisión a distancia de los datos e informaciones se realiza básicamente de dos maneras (ver la figura de la página siguiente):

- **transmisión asíncrona**
- **transmisión síncrona**

El modo asíncrono se define precisamente con START-STOP, puesto que los bits que contienen la información propiamente dicha (carácter) van precedidos y seguidos de dos bits especiales: el **bit de start**, el cual avisa que está llegando un carácter, y el **bit de stop**, el cual informa que el carácter se ha terminado.
El modo síncrono no requiere el uso de los bits de start y de stop, pero emplea caracteres de sincronismo posicionados en la cabeza y la cola de los «grupos» de caracteres a transmitir. En este método, la recepción de un número predeterminado de caracteres de sincronismo informa al receptor que está iniciando un texto, que se cerrará con otros caracteres de sincronismo (además de los caracteres especiales de final de texto).



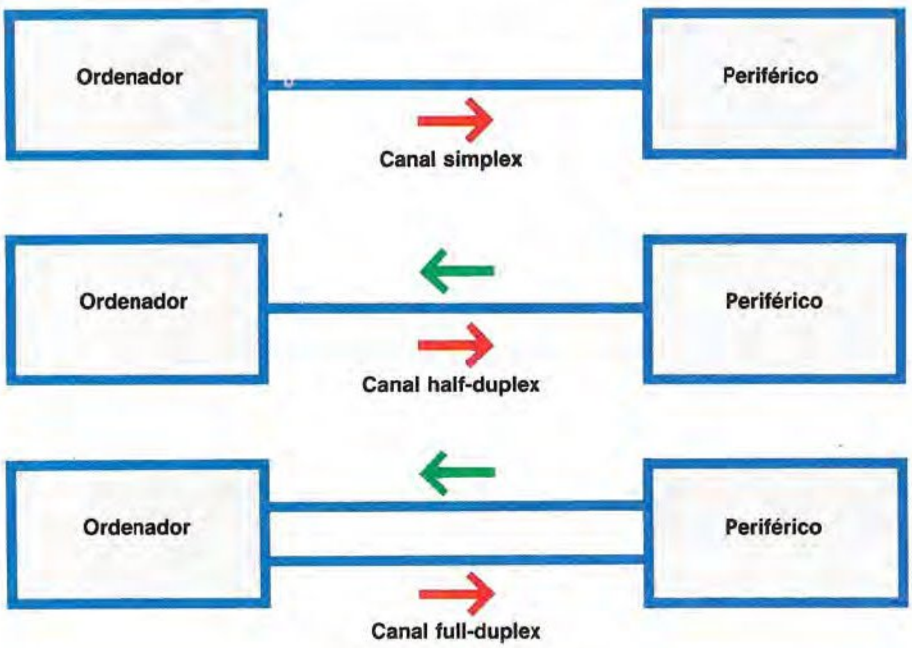
Canales simplex, half-duplex, full-duplex

Como se ha dicho, el dispositivo físico utilizado para una conexión se llama **línea de comunicación**; si la conexión se realiza a distancias cortas la línea coincide físicamente con el cable de conexión y con los interfaces del ordenador y del periférico. Para las conexiones remotas, la línea más utilizada es la telefónica, dotada de instrumentos especiales que se ilustrarán más adelante. Establecida una línea, se define como **canal** el camino a través del cual fluye la información. El concepto de canal nace porque en una línea es posible hacer viajar varios tipos de información y, por tanto, se tiene necesidad de seleccionar el camino adecuado.
Se distinguen tres tipos de canales:

- **simplex**
- **half-duplex**
- **full-duplex**

El canal simplex permite que las informaciones fluyan sólo en un sentido: siempre desde el ordenador al periférico, o siempre desde el periférico al ordenador si el periférico es de sólo entrada. El canal simplex suele utilizarse poco, porque en casi todas las conexiones ordenador-periférico es necesario que las informaciones viajen en ambos sentidos: una impresora, por ejemplo, debe informar al ordenador que ha encontrado un error en el buffer que contiene los caracteres a imprimir. El canal simplex puede compararse a una carretera de sentido único. En un canal half-duplex, en cambio, las informaciones pueden viajar en ambos sentidos, pero no simultáneamente; esto significa que, en un determinado momento, el canal sólo es activo en un sentido de comunicación y se invierte cada vez que cambia a transmisión o a recepción. Por tanto, el canal half-duplex puede compararse a una carretera de circulación de sentido único alternado, controlada por dos semáforos posicionados en los extremos de dicha carretera. En el canal full-duplex, las informaciones viajan

TIPOS DE CANAL DE COMUNICACION



simultáneamente en ambos sentidos, y sin necesidad de interrupción. Esto es posible sólo si el canal está constituido físicamente por dos cables diferentes, uno para cada sentido de transmisión.

Ahora planteamos una importante pregunta: dado un canal y una línea de comunicación, ¿qué cantidad de informaciones es posible transmitir entre los dos dispositivos en un segundo? La respuesta depende del tipo de conexión, de la modalidad de comunicación y del tipo de canal.

La unidad de medida de la velocidad de transmisión es el bit por segundo (sigla bps). El número máximo de bps expresa la capacidad de transmitir datos en un canal o en una línea. He aquí algunos órdenes de magnitud:

- conexiones paralelas: hasta algunos millones de bps
- conexiones serie: algunos centenares de miles de bps en modalidad síncrona; algunas decenas de miles de bps en modalidad asíncrona.

Comunicaciones por línea telefónica

Las conexiones examinadas hasta ahora son típicas de las distancias cortas entre ordenador y periférico, soliéndose entender por distancia corta un recorrido no superior a 30 o 40 m. Para distancias mayores se plantean problemas ligados a la propagación de las señales, y es necesario recurrir a las líneas telefónicas.

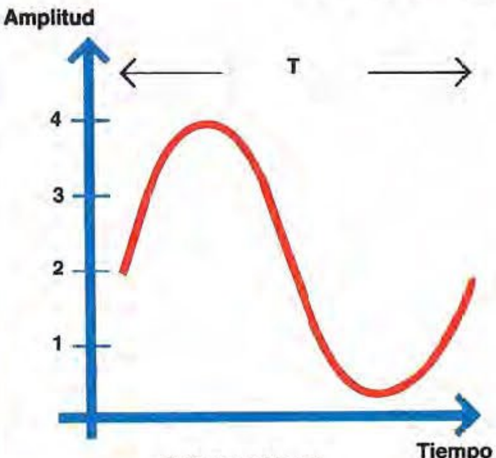
Características de las señales

Las señales empleadas en las telecomunicaciones son **periódicas**, es decir, se repiten transcurrido un cierto tiempo T (período si se mide en segundos). Las señales procesadas en el ordenador son de tipo digital, mientras que las otras señales empleadas para comunicar informaciones son de tipo analógico.

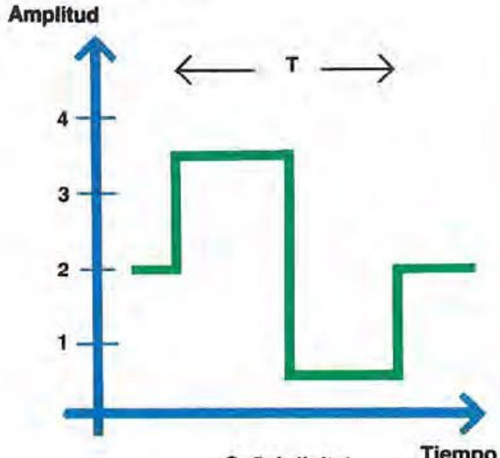
Una señal periódica está definida por tres parámetros:

- amplitud
- frecuencia
- fase

TIPOS DE SEÑALES PERIODICAS



Señal analógica



Señal digital

La amplitud de la señal está representada por el valor máximo que ésta puede asumir. En una señal analógica, el nivel varía continuamente para asumir los mismos valores después de cada período T. El nivel de una señal digital no varía continuamente, sino que permanece constante durante un cierto tiempo y después cambia bruscamente. En la figura de arriba, el nivel de la señal digital es constante para T/2 y vale 3.5, para pasar después al valor 0.5.

La frecuencia indica las veces que la señal se repite en un segundo: una señal que se repite 5 veces en un segundo tiene una frecuencia de 5 Hz; su período será igual a 1/5 de segundo (el período es el inverso de la frecuencia).

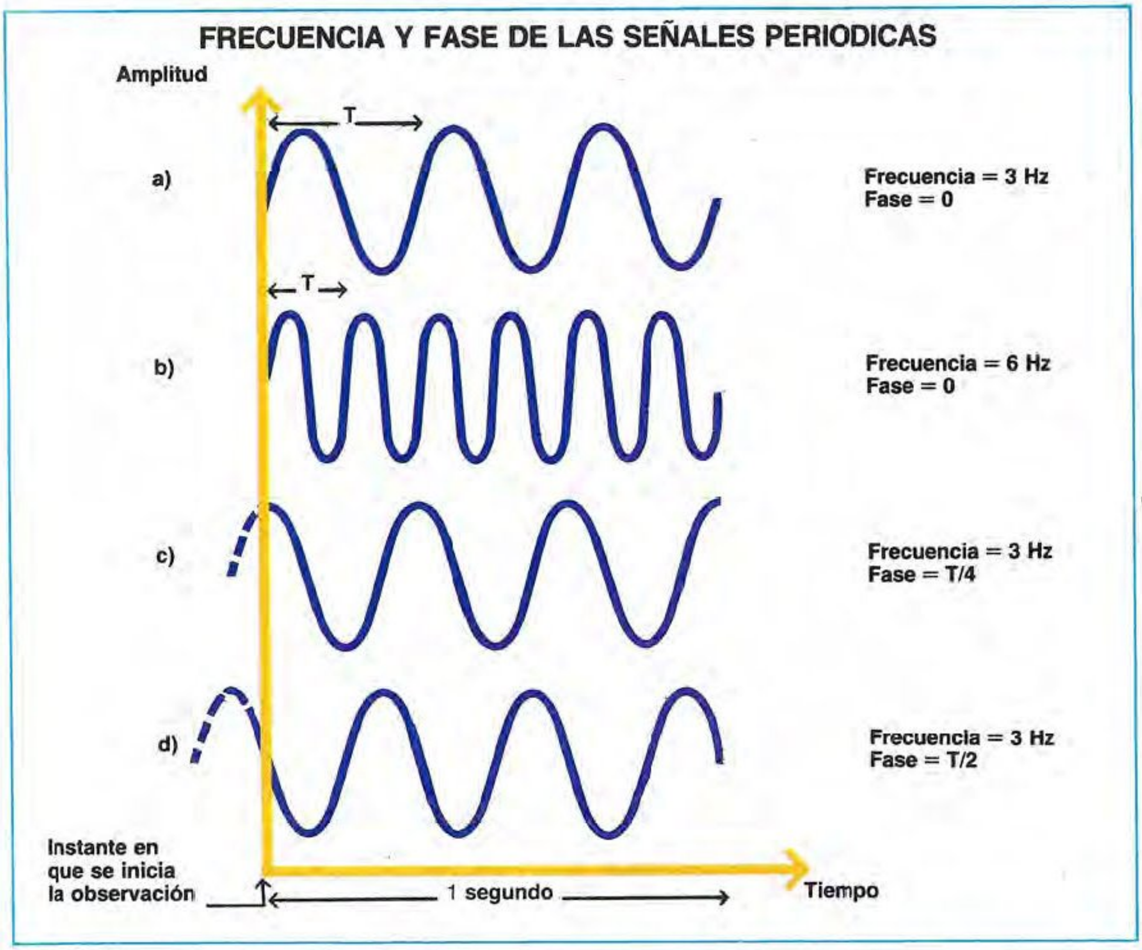
La fase representa en cierto modo el retardo o el adelanto de la señal con respecto al instante en que se inicia la observación, y se mide en fracciones de período.

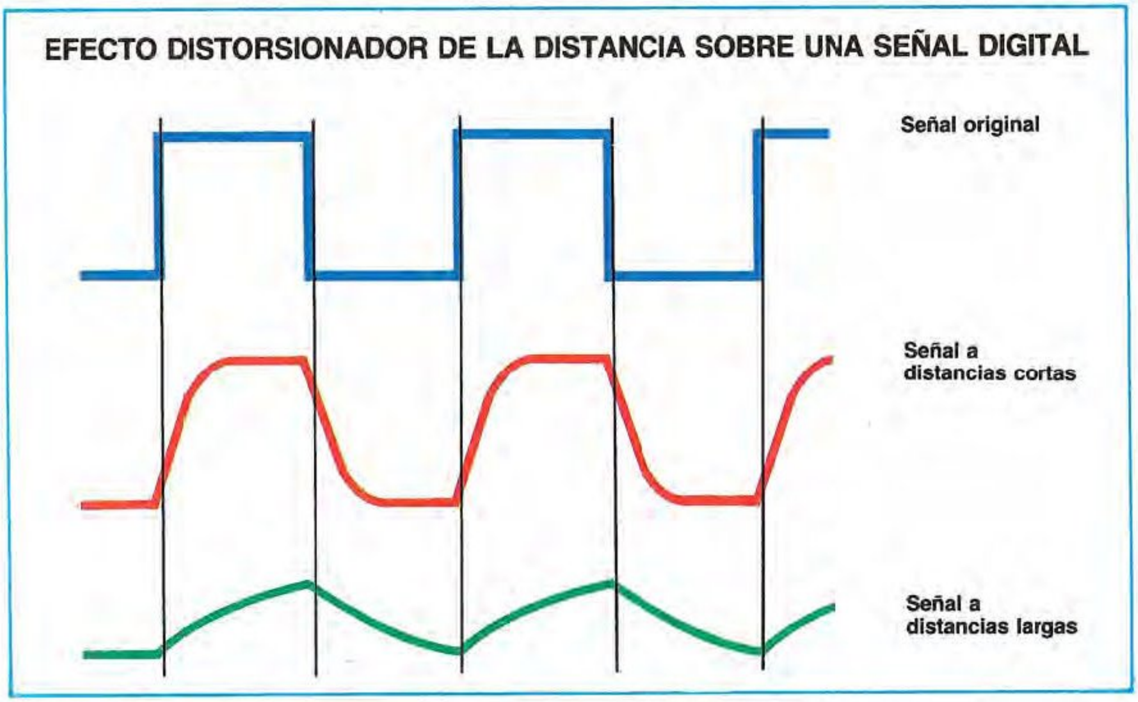
En la figura de arriba de la página siguiente se han representado algunas señales que aclaran los conceptos de frecuencia y fase. La señal c se dice que está desfasada T/4 en adelanto, porque es como si se iniciase un cuarto de período antes del instante en que se inicia la observación de las señales a y b; análogamente, la señal d está desfasada T/2 en adelanto, puesto que es como si se iniciase medio período antes.

Una señal digital que se propaga sobre una línea sufre distorsiones de forma cuya importancia se incrementa con el aumento de las distancias recorridas (ver la figura de abajo de la pág. siguiente). Por tanto, existen distancias límite (30 a 40 m) más allá de las cuales la señal no puede reconocerse con seguridad. Dada una línea de comunicación (por ejemplo un cable), su capacidad de transmitir informaciones es proporcional a la **anchura de banda en frecuencia**. Con este término se entiende la diferencia en herzios entre el valor máximo y el valor mínimo de las frecuencias que corresponden a señales no distorsionadas. Por ejemplo, decir que la banda de una línea va de 300 a 3000 Hz significa que las señales que tienen una frecuencia comprendida entre estos valores no están distorsionadas (excesivamente). Extrapolando el concepto puede afirmarse que cuanto más ancha es la banda de una línea, tanto menos distorsionada es la señal.

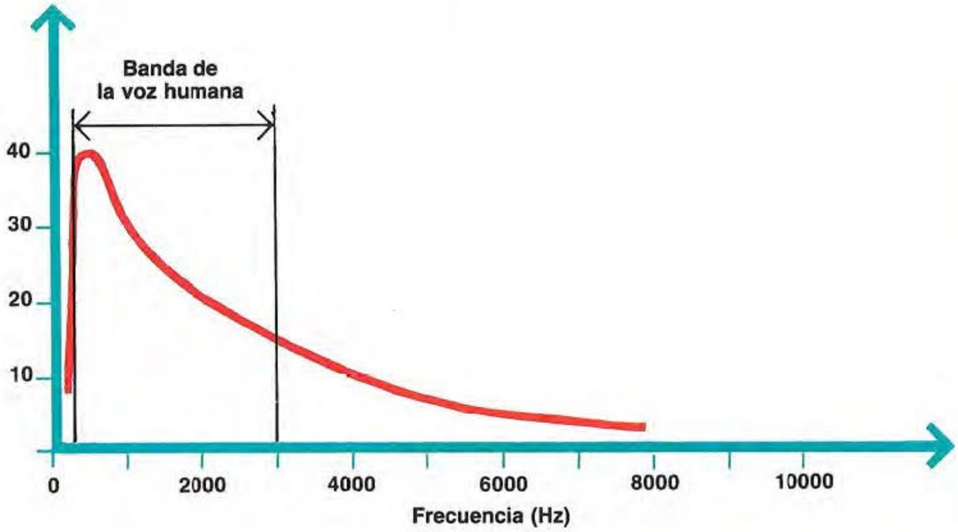
Para los cables se adoptan adecuadas disposiciones de construcción (por ejemplo cables apantallados) con lo que los 30 o 40 m de longitud máxima de una línea para señales digitales pueden ampliarse a 300 o 400 m. En cualquier caso, la conexión de un periférico situado en Sevilla a un ordenador situado en Bilbao no puede resolverse de esta manera. En estos casos debe utilizarse una línea telefónica.

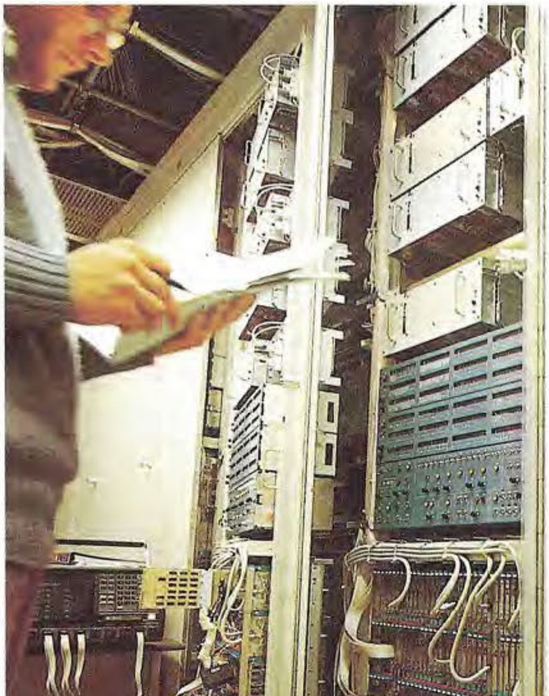
La estructura de las líneas telefónicas se optimiza en función de la necesidad de difusión de la voz humana y, por tanto, de señales que tengan frecuencias comprendidas entre 300 y 3000 Hz; esta anchura de banda es suficiente para que la voz de una persona llegue a su destino clara y reconocible (figura de la pág. 1324). Por tanto, las líneas telefónicas pueden transmitir informaciones, pero las señales digitales, dada su diferente estructura a la de las señales que transmi-


FRECUENCIA Y FASE DE LAS SEÑALES PERIODICAS
Amplitud
T
a)
Frecuencia = 3 Hz
Fase = 0
T
b)
Frecuencia = 6 Hz
Fase = 0
c)
Frecuencia = 3 Hz
Fase = T/4
d)
Frecuencia = 3 Hz
Fase = T/2
Instante en
que se inicia
la observación
1 segundo
Tiempo


EFECTO DISTORSIONADOR DE LA DISTANCIA SOBRE UNA SEÑAL DIGITAL
Señal original
Señal a
distancias cortas
Señal a
distancias largas

ATENUACION DE LA VOZ EN UNA LINEA TELEFONICA





Central electrónica Italtel para los servicios telex y de datos.

te la voz humana, no podrían alcanzar estas distancias elevadas. Por tanto, la señal digital debe transformarse en una señal analógica con las frecuencias incluidas en la banda que puede garantizar la línea telefónica. Esto obliga a utilizar determinados instrumentos de conversión de la señal antes de que ésta se introduzca en la línea telefónica y en la recepción; estos instrumentos son los **modems**, y su uso permite alejar un periférico a cualquier distancia.

Las líneas telefónicas empleadas en la teletransmisión de datos pueden clasificarse en dos tipos:

- líneas conmutadas
- líneas dedicadas

Las líneas conmutadas se emplean como los teléfonos normales: en un aparato telefónico se marca un número al que responde el ordenador con que se quiere conectar, y de esta manera se establece la conexión.

En cambio, las líneas dedicadas establecen una conexión permanente entre dos puntos; en este caso no es necesario marcar números.

Técnicas de modulación. Los modems

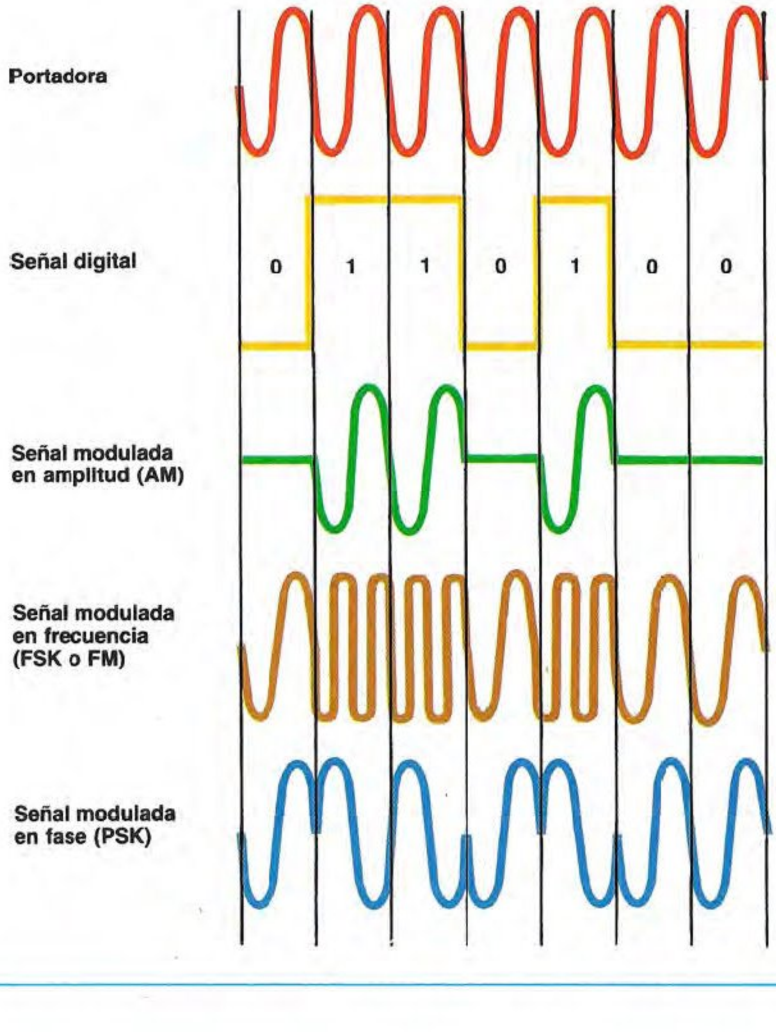
Las transformaciones digital-analógica y analógica-digital se indican con los términos modulación y demodulación, y se realizan mediante los modems. La palabra modem es la contracción

de MOdulación y DEModulación. La modulación es la transformación de una señal digital en una señal analógica, cuya banda de frecuencias esté dentro de la que puede propagar una línea telefónica sin distorsión. En cambio, la demodulación es la recuperación de la señal digital original.
La modulación puede realizarse utilizando diversas técnicas (ver la figura de abajo), y se habla según el caso de

- modulación de amplitud
- modulación de frecuencia
- modulación de fase

En estos tres casos se hace uso de una señal de referencia analógica llamada **portadora**, generada por un circuito presente en el modem. La modulación de amplitud hace que en la línea haya portadora de acuerdo con un estado lógico (por ejemplo, el estado 1) y no haya portadora en correspondencia con el otro estado lógico (el estado 0).
Como puede observarse en la figura, la transmisión de dos estados lógicos 1 consecutivos comporta la presencia de la portadora en la línea durante un tiempo más largo que el que se tiene para la transmisión de un solo estado lógico 1. Esto significa que para detectar correctamente los momentos que corresponden a los estados lógicos debe emplearse un circuito-reloj (clock). La modulación de amplitud acostumbra a denominarse con la sigla AM (Amplitude Modulation).

TECNICAS DE MODULACION



La modulación de frecuencia se basa en la presencia de dos frecuencias en la línea: la frecuencia de la portadora para un estado lógico (por ejemplo para el estado 0) y una frecuencia múltiplo de la portadora para el otro estado lógico (el estado 1). La amplitud de la señal es la misma para ambas frecuencias, y por tanto para los dos estados digitales que reproducen. La modulación de frecuencia acostumbra a denominarse con la sigla FSK (Frequency Shift Key) o FM (Frequency Modulation).

La modulación de fase deja constantes en la línea tanto la amplitud como la frecuencia de la señal. La transmisión de la información se basa en la variación de fase de T/2. La figura de abajo aclara el significado de una señal con fase 0 y de una señal desfasada T/2.

La demodulación, o sea la reconstrucción de la señal digital original, emplea circuitos que reproducen la señal de manera diferente según la técnica de modulación adoptada. Normalmente, en un modem hay tanto los circuitos que pueden enviar a la línea señales moduladas como los que pueden reconstruir las señales digitales a la recepción de las señales moduladas. Los modems instalados en los dos extremos de una línea deben ser compatibles, o sea deben trabajar a la misma velocidad y según las mismas técnicas de modulación/demodulación. De hecho existen modems que, si bien trabajan a la misma velocidad, adoptan técnicas de modulación/demodulación diferentes; estos modems se definen **equivalentes**. Los modems se distinguen también, según la modalidad de transmisión, en síncronos y asíncronos. Normalmente, las velocidades de conmutación más elevadas se obtienen con los modems síncronos.

Una última distinción debe hacerse sobre la banda en que pueden trabajar los modems y, para este caso, se tiene:

— modems que trabajan por debajo de la banda vocal (hasta 300 Hz)
— modems en la banda local (300 a 3000 Hz)
— modems de banda ancha (por encima de 3000 Hz)

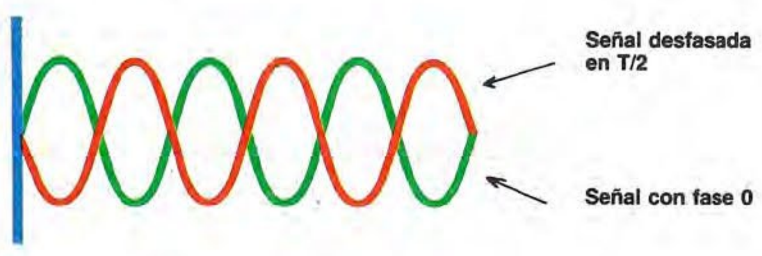
En el primer caso, la velocidad de transmisión no va más allá de 150 bps; en el segundo se tienen velocidades de 1200 a 2400 bps para los modems asíncronos y de 2400 a 19200 bps para los modems síncronos. En el tercer caso se obtienen velocidades superiores, pero es necesario emplear técnicas de modulación muy sofisticadas, y, por tanto, con costes más elevados. La clasificación indicada se esquematiza en el gráfico superior de la página siguiente.

Configuraciones de las conexiones

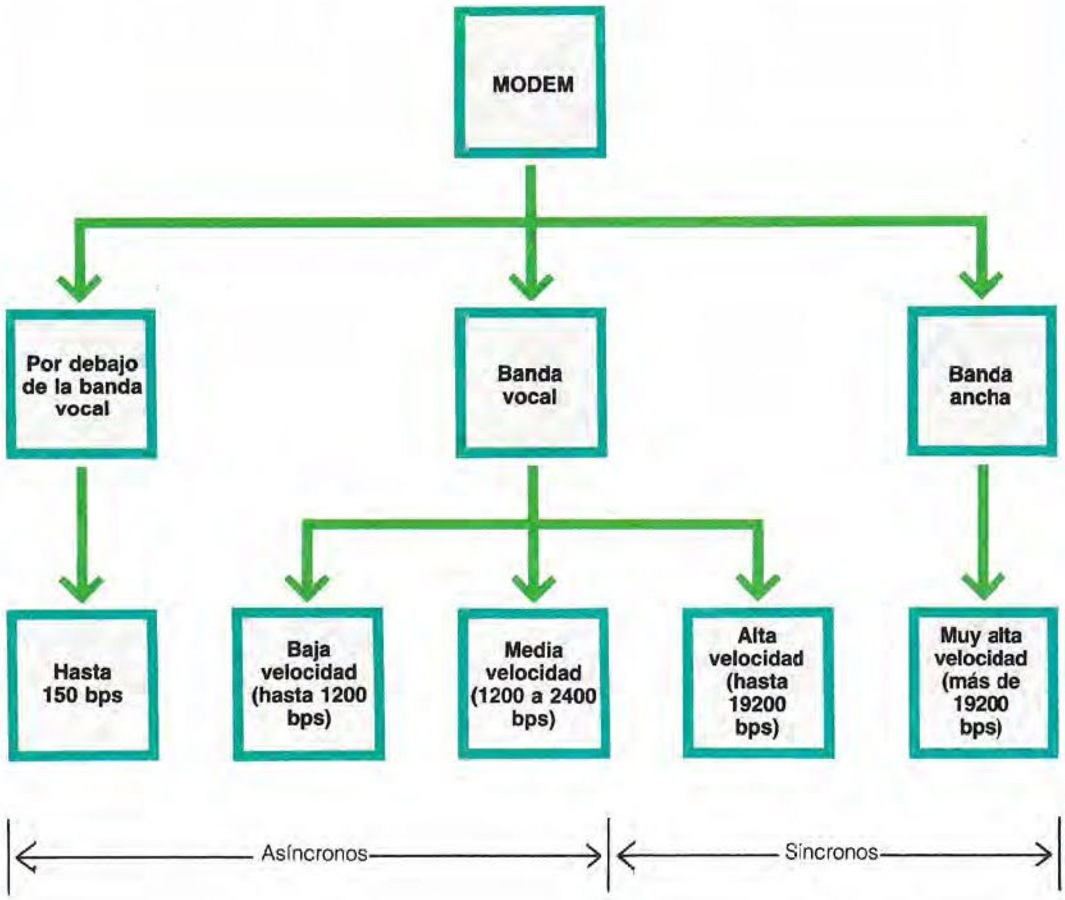
La clasificación de las conexiones que ahora describiremos sirve tanto para las distancias cortas como para las largas, o sea con y sin modems. Para efectuar esta generalización debe introducirse una terminología particular. Los aparatos que deben conectarse entre sí se definen con las siglas DTE, Data Terminal Equipment, tanto si se trata de un ordenador como de terminales, etc. En cambio, los dispositivos empleados para la conexión se definen con la sigla DCE, Data Communication Equipment.

Esta terminología se emplea en el gráfico de abajo de la página siguiente, donde se presenta el primer tipo de configuración, o sea la conexión directa de un periférico al ordenador.

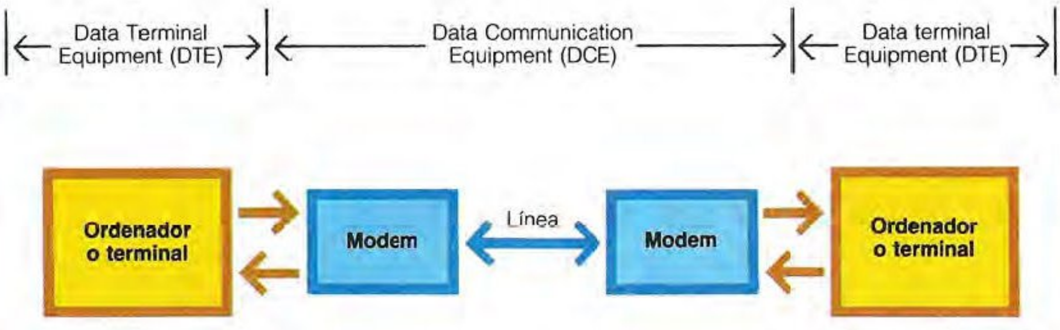
EVIDENCIA DE LA FASE



CLASIFICACION DE LOS MODEMS



CONFIGURACION PUNTO A PUNTO



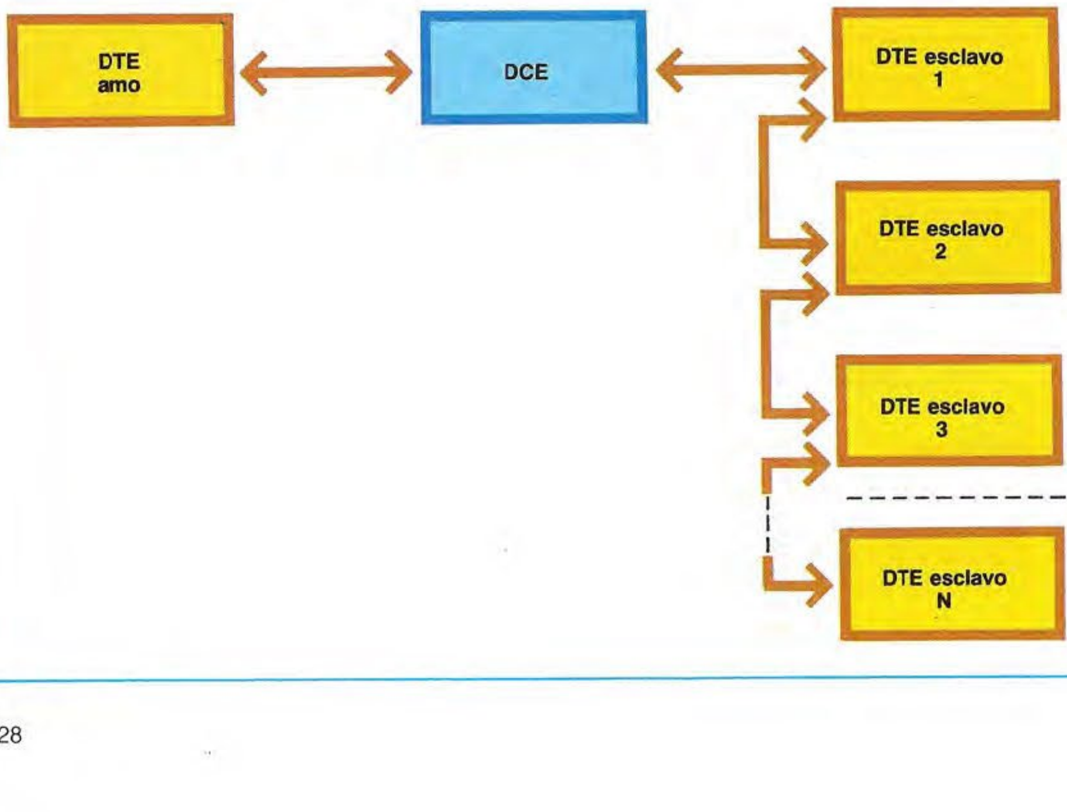
Configuración punto a punto. La configuración esquematizada en la figura de la página anterior (abajo) se llama de **punto a punto** y emplea una línea de comunicación para cada par de DTE. Para conectar otro periférico DTE al ordenador en modalidad de punto a punto deberá emplearse otra línea (otro DCE). La configuración punto a punto es sencilla de realizar y tiene la ventaja de tener siempre la disponibilidad de la línea; en cambio, tiene la desventaja de que es necesario emplear tantas líneas como periféricos tengan que conectarse, con el consiguiente aumento de los costos, especialmente en el caso de comunicaciones de larga distancia. En esta última eventualidad si la velocidad de transmisión no es elevada, pueden emplearse líneas conmutadas, pero en cada caso es necesario un par de modems por línea.

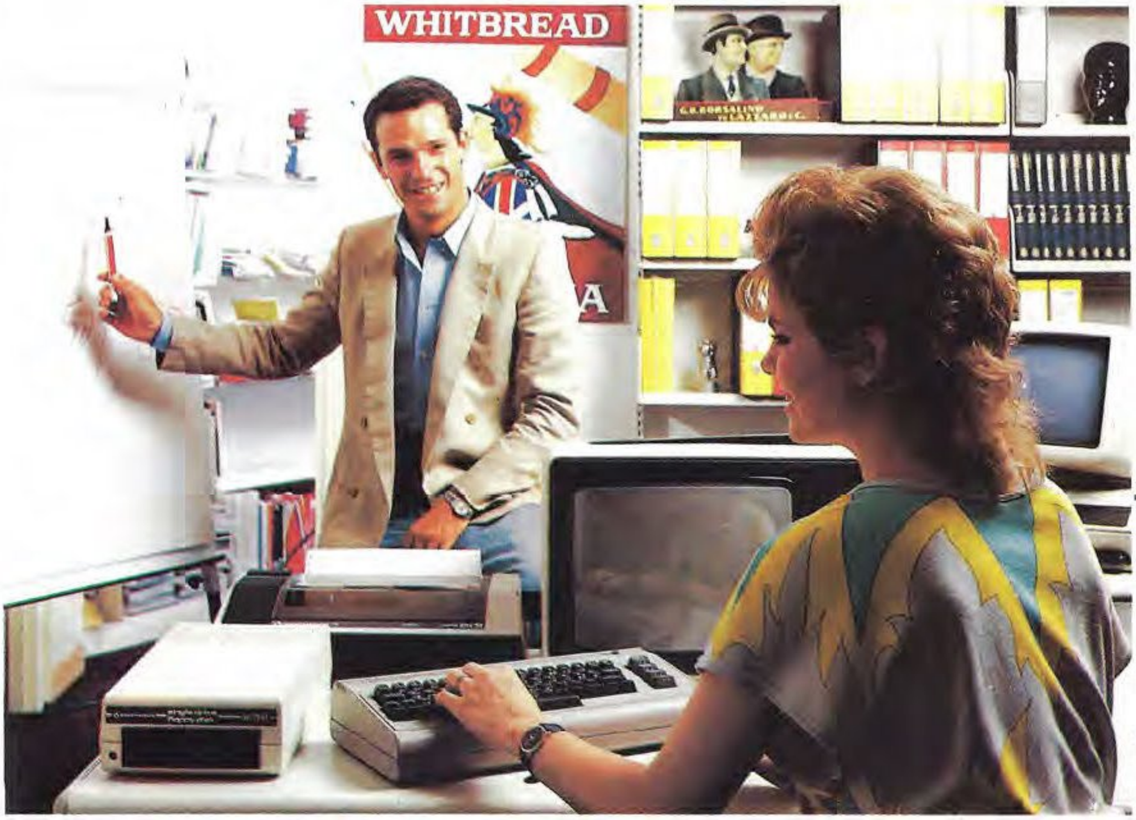
Configuración multipunto. Una configuración que reduce los costos de la línea es la denominada **multipunto**, esquematizada en la figura de abajo, donde la línea de comunicación y los modems se han compendiado en el bloque DCE, y los elementos a conectar, tanto si son terminales como el ordenador, se identifican con el bloque DTE. En un extremo de la línea hay un tipo particular de DTE, el **DTE amo**, que controla el diálogo en la línea; en el otro extremo hay N dispositivos definidos como **DTE esclavo**, todos conectados a la misma línea, que obedecen a las peticiones del DTE amo. Cada DTE esclavo está dotado de un código de reconocimiento (en la figura de abajo los números 1, 2, 3,..., N) que el amo utiliza para saber si se está transmitiendo en cierto momento por la línea, o bien para identificar al que debe recibir. El coloquio en una configuración multipunto está gestionado por dos actividades diferentes, ambas controladas por el DTE amo:

- polling = invitación a transmitir un mensaje
- selection = invitación a recibir un mensaje

En la fase de polling, el DTE amo interroga cada vez a uno de los DTE esclavos, verificando si cada uno de ellos tiene algo para transmitir. Si un esclavo tiene un mensaje que enviar en la línea, puede hacerlo sólo cuando es interrogado por el amo; en cambio, si un esclavo no tiene nada para enviar, el amo pasa a interrogar al siguiente esclavo, y cuando llega al último vuelve al primero. Por lo tanto, la fase de polling es cíclica.

CONFIGURACION MULTIPUNTO



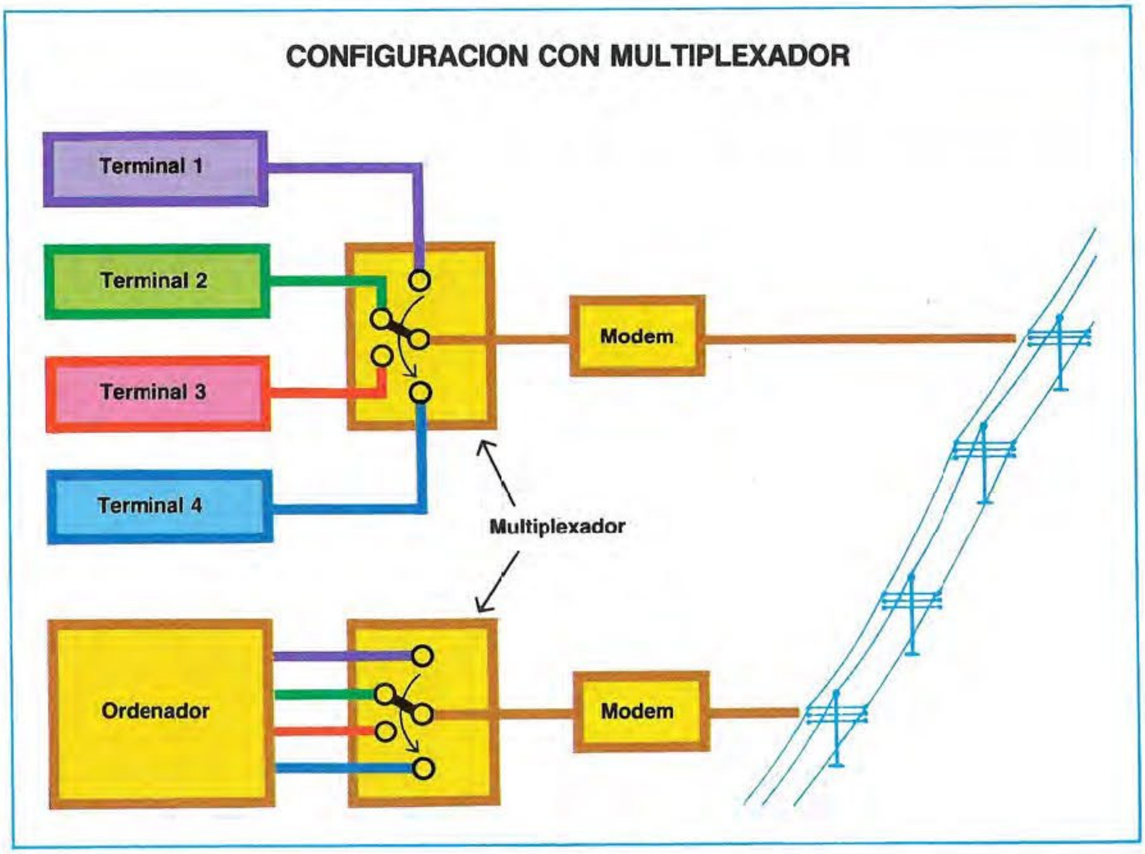


El sistema de proceso Commodore 64 en su configuración completa.

Si un esclavo interrogado envía un mensaje, el aparato que transmite es reconocido por el receptor mediante el código de identificación del DTE transmisor, incluido en el propio mensaje. En la fase selection, el DTE amo se dirige a un esclavo informándolo de un mensaje de llegada, y el esclavo se predispone para recibirlo. Consideremos un ejemplo. Suponiendo que el amo sea un ordenador y que los esclavos sean N terminales vídeo, el ordenador estará dotado de un interfaz que puede controlar el coloquio en modalidad multipunto, y lo mismo deberá suceder para los terminales vídeo. Del ordenador parte un cable al cual están conectados todos los terminales. El ordenador realiza cíclicamente la fase polling en la que son habilitados los terminales, uno cada vez, que deben transmitir. Cuando un terminal transmite, el ordenador reconoce el terminal por su código y adquiere los datos transmitidos por éste. Suponiendo que estos datos deban presentarse en otro terminal, perteneciente a la misma cadena multipunto, el ordenador activa la fase selection y abre sobre la línea el buffer de datos en que hay el código del dispositivo que debe recibir. El receptor adquiere entonces los datos y los presenta. Como puede observarse, en la configuración multipunto, el diálogo entre los DTE no es completamente asíncrono, y esto plantea problemas de espera durante el uso de la línea. En este ejemplo otro terminal podrá transmitir sólo cuando se ha terminado la fase selection: si el buffer de datos a presentar es largo, será largo también el tiempo de espera del terminal que quiere transmitir. Por tanto, la línea se convierte en el «cuello de botella» del sistema, y esto implica la necesidad de encontrar un buen compromiso entre el número de DTE a conectar y la cantidad de información a transmitir por unidad de tiempo.

Configuraciones de multiplexador. Una configuración que minimiza el número de líneas y plantea menos problemas al multipunto es el **multiplexador**, y se emplea cuando se dispone de un canal de datos de alta velocidad y de varios DTE que deben comunicarse.

Por ejemplo, consideremos un ordenador que debe comunicar con cuatro terminales utilizando una sola línea (ver la fig. de la pág. siguiente). El multiplexador puede verse como un ins-



trumento que permite evitar la confusión de las señales presentes en la misma línea. Esto puede obtenerse con métodos diferentes.

El **multiplexador de división de tiempo** es un sistema que concede un intervalo fijo de tiempo a cada terminal que debe transmitir y reproduce con el mismo tiempo las informaciones al otro extremo de la línea. En el ejemplo de cuatro líneas de la figura de arriba se tendrá la temporización detallada en la figura de arriba de la página siguiente, donde en el instante t_1, la línea 1 envía el carácter A, en el instante t_2 la línea cuatro envía el carácter B, y así sucesivamente.

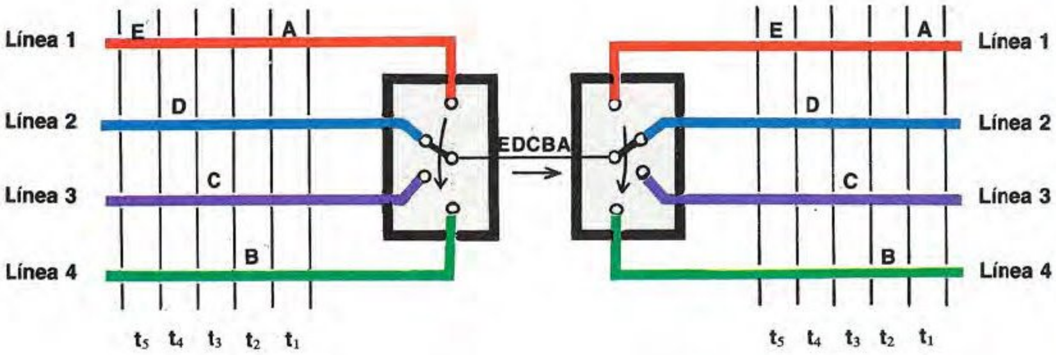
Con el **multiplexador de división de frecuencia**, cada canal de datos correspondiente a cada terminal trabaja en una determinada banda de frecuencias. El principio es similar a la transmisión de radio: hay varias emisoras que transmiten simultáneamente en frecuencias diferentes, y un aparato de radio receptor sólo capta la emisora que sintoniza (ver la fig. del centro de la pág. siguiente). En el ejemplo ilustrado arriba, cada terminal tiene a su disposición una banda precisa de frecuencias por la que transmite, y el ordenador tiene cuatro puntos de sintonía. Puede observarse que el multiplexador de división de tiempo puede imaginarse como un interruptor de alta velocidad y, por tanto, la capacidad de transmitir informaciones queda íntimamente ligada a las características físicas del instrumento. Los parámetros indicadores de esta capacidad son los que proporcionan los bits por segundo (bps) totales y el número de puertas disponible. Por ejemplo, un multiplexador de 38400 bps con 16 puertas puede hacer comunicar 16 terminales de 2400 baudios* cada uno. Si los terminales están conectados a distancia, la línea telefónica y los modems empleados deben sostener los 38400 bps. La capacidad de transmisión de un multiplexador de división de frecuencia se mide en términos de banda de frecuencia total: cuanto más amplia es la banda, tantos más dispositivos podrán comunicar.

Interfaz de comunicación

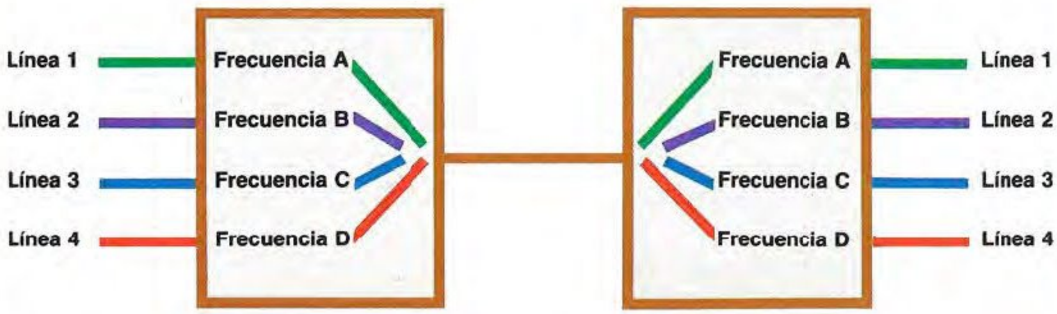
La terminología DTE/DCE debe completarse

* El baudio expresa la velocidad de transmisión en bits por segundo. El nombre de la unidad de medida se deriva del de Baudot, un pionero de la informática.

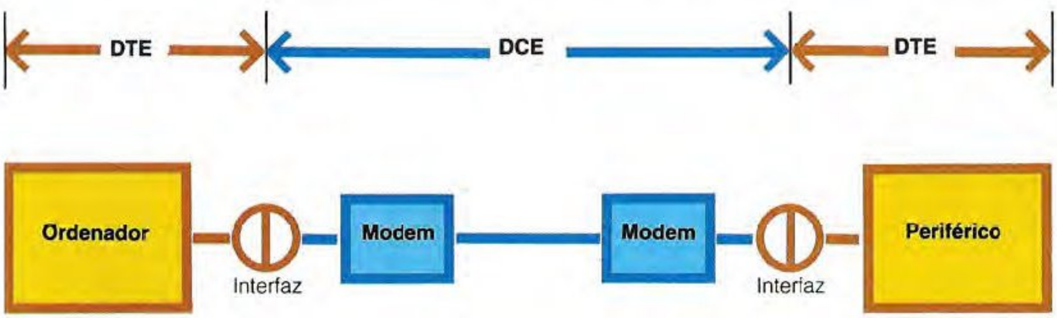
MULTIPLEXADOR DE DIVISION DE TIEMPO



MULTIPLEXADOR DE DIVISION DE FRECUENCIA



CONEXION PUNTO A PUNTO CON EVIDENCIA DE LOS INTERFACES



con la introducción del concepto de **interfaz**. En la teletransmisión de datos, se entiende con este término un dispositivo que permite la conexión y el coloquio entre los DTE y los DCE (ver la figura de abajo de la pág. anterior). La presencia de los interfaces de comunicación se hace principalmente para estandarizar las conexiones.
Las normas que regulan la transmisión de datos vía modem han sido emitidas por dos entidades, el CCITT (Comité Consultivo Internacional para Telefonía y Telegrafía) y la EIA (Electronics Industries Association).
Las normas CCITT tienen validez para Europa, mientras que las normas EIA valen para Estados Unidos, pero desde el punto de vista práctico, ambas recomendaciones son equivalentes. Las normas para las transmisiones vía modem más difundidas llevan los nombres (protocolos)

EIA-RS232C
CCITT-V24

y establecen reglas mecánicas, eléctricas y de circuitos muy precisas.
La parte del circuito de las normas corresponde a la gestión de las señales a lo largo de la línea de comunicación.

La tabla de abajo contiene la nomenclatura de los circuitos según la EIA y el CCITT (se habla indistintamente de señal o de circuito).
El uso de algunas señales y no de otras está ligado al tipo de comunicación utilizado (síncrona, asíncrona, half-duplex, full-duplex) y, por tanto, al tipo de modem utilizado. Por tanto, según el tipo de comunicación, las señales se subdividen en cuatro grupos:

1 / referencia	(101 y 102)
2 / datos	(103 y 104)
3 / control	(105, 106, 107, 108.2, 109, 125)
4 / temporización	(114, 115)

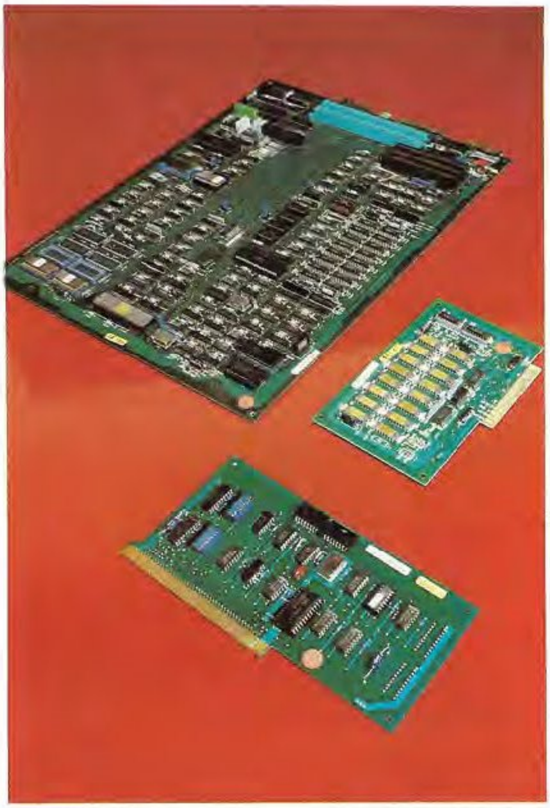
Describiremos los que están indicados al lado de cada grupo, puesto que son los más utilizados en la mayor parte de las conexiones.

- Circuito 101 (AA): Protective Ground. Es la señal de tierra conectada al chasis de los aparatos.
- Circuito 102 (AB): Signal Ground. Es la referencia para todas las señales emitidas por un DTE y un DCE (excluida la señal 101). Los circuitos 101 y 102 a veces se conectan entre sí para reducir los parásitos.

RECOMENDACIONES EIA-RS232C CCITT-V24

Patilla	Hacia DTE DCE	Circuito RS232C	Circuito V24	Descripción
1		AA	101	Protective Ground
2	→	BA	103	Transmitted Data
3	←	BB	104	Received Data
4	→	CA	105	Request to Send
5	←	CB	106	Clear to Send
6	←	CC	107	Data Set Ready
7	←	AB	102	Signal Ground
8	←	CF	109	Detector for Signal Line Received (Carrier Detector)
9		–	–	
10		–	–	
11		–	–	
12	←	SCF	122	Detector for Secondary Signal Line Received
13	←	SCB	121	Secondary Clear to Send
14	→	SBA	118	Secondary Transmitted Data
15	←	DB	114	Transmission of Signal Timing
16	←	SBB	119	Secondary Received Data
17	←	DD	115	Receive of Signal Timing
18		–	–	
19	→	SCA	120	Secondary Request to Send
20	→	CD	108.2	Data Terminal Ready
21	←	CG	110	Signal Quality Detector
22	←	CE	125	Ring Indicator
23	→	CH/CI	111/112	Selector of Data Signal Rate
24	→	DA	113	Transmission of Signal Timing
25		–	–	

- Circuito 103 (BA): Transmitted Data (de DTE a DCE). Contiene los datos a transmitir en la línea. Este circuito no puede trabajar si no están activados los circuitos 105 (Request to Send), 106 (Clear to Send), 107 (Data Set Ready) y 108.2 (Data Terminal Ready).
- Circuito 104 (BB): Received Data (de DCE a DTE). Contiene los datos a enviar al DTE recibidos de la línea. La condición de ON de este circuito está ligada a la activación del circuito 109 (Carrier Detector): si éste está en OFF, el circuito 104 no puede trabajar.
- Circuito 105 (CA): Request to Send (de DTE a DCE). Indica que el DTE tiene necesidad de transmitir, y predispone al DCE para la transmisión (el modem emite la portadora).
- Circuito 106 (CB): Clear to Send (de DCE a DTE). Indica que el DCE está preparado para transmitir. El DCE pone la señal en ON después de que éste ha recibido el Request to Send. En la práctica, el Clear to Send realiza el proceso del Request to Send, después del retardo inicial necesario para que el DCE se predisponga.
- Circuito 107 (CC): Data Set Ready (de DCE a DTE). La condición de ON indica que el DCE está conectado a la línea y preparado tanto para transmitir como para recibir. La emisión de las señales en la línea está ligada a la condición de ON del Data Set Ready: la única señal que puede operar con Data Set Ready OFF es el 125 (Ring Indicator).
- Circuito 108.2 (CD): Data Terminal Ready (de DTE a DCE). Cuando está en ON, el DTE informa al DCE que está preparado tanto para transmitir como para recibir. Inmediatamente después de que el DTE ha alzado el Data Terminal Ready, el DCE alza el Data Set Ready; por tanto, las dos señales tienen el mismo proceso.
- Circuito 109 (CF): Carrier Detector (de DCE a DTE). Indica que la señal presente en la línea está dentro de los niveles útiles (control de nivel). Si el Carrier Detector está en condición OFF ya no es posible recibir señales de la línea.
- Circuito 125 (CE): Ring Indicator (de DCE a DTE). En condición ON señala la llegada de una llamada y da inicio a la conexión.
- Circuito 114 (DB): Transmission of Signal Timing (de DCE a DTE). Lo utiliza el DCE (modem) para temporizar el DTE (ordenador o terminal) en transmisión.



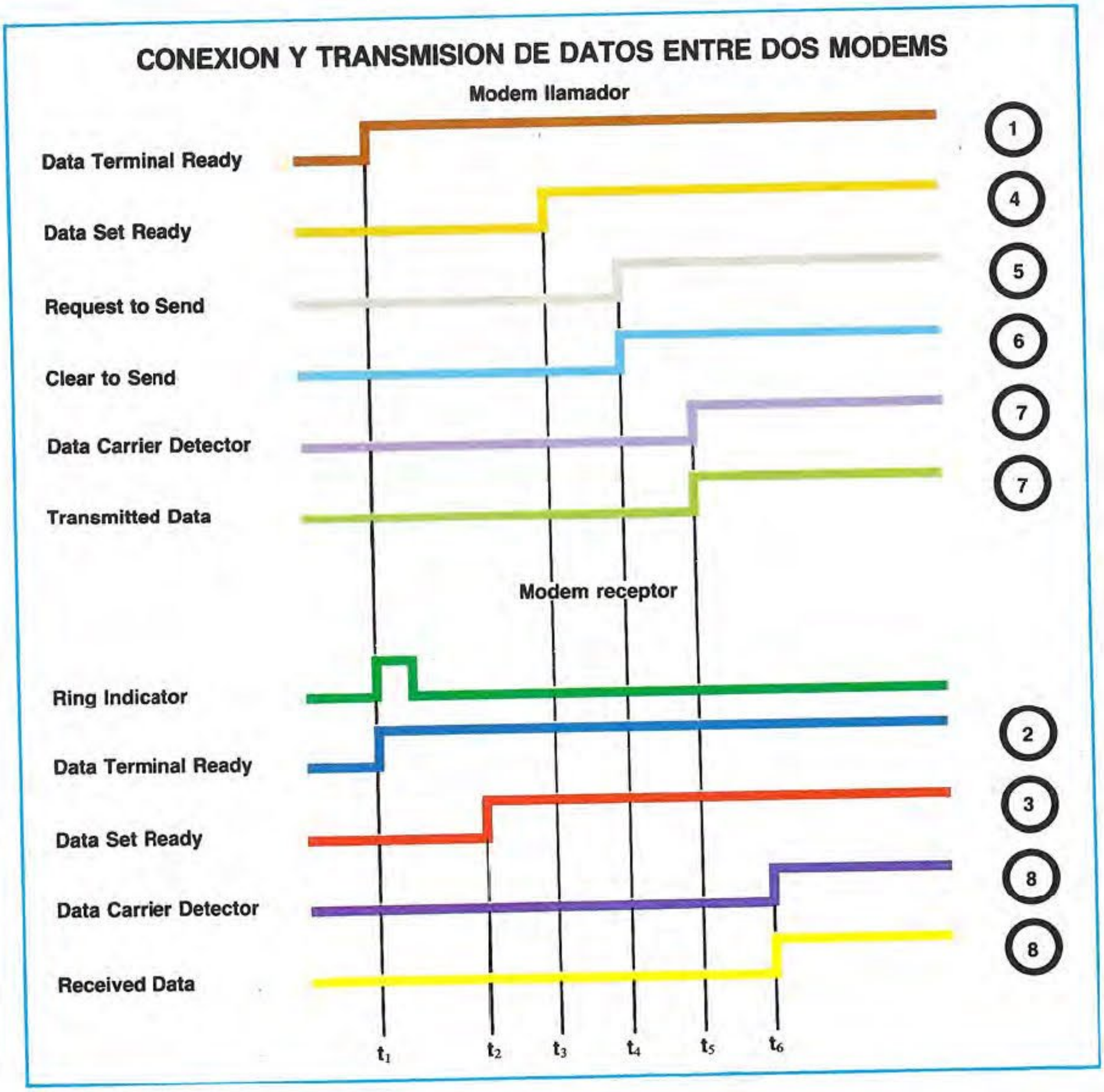
La tarjeta madre y dos placas de circuito de un sistema Olivetti.

- Circuito 115 (DD): Receive of Signal Timing (de DCE a DTE). Lo utiliza el DCE (modem) para temporizar el DTE durante la recepción. La condición ON del circuito 115 está ligada al circuito 109 (Data Carrier Detector); si el 109 está en OFF, también el 115 está en OFF.

Como ejemplo sobre el uso de las señales descritas examinaremos la secuencia de conexión y transmisión de datos entre dos modems distantes y funcionando sobre una línea telefónica conmutada.
La secuencia temporal de las fases sencillas se esquematiza en la figura de la página siguiente.

1 / En el modem llamador, la señal Data Terminal Ready es activa y un operador marca el número telefónico que corresponde al modem receptor.

2 / El modem receptor activa la Ring Indicator; también en el lado receptor, la Data Terminal Ready está en ON, señal de que el DTE está preparado (terminal en marcha).



3 / En el instante t_2, el modem receptor activa la Data Set Ready y el operador que llama recibe en escucha una señal acústica que indica la realización de la conexión.

4 / El operador que llama aprieta un pulsador de su modem y reengancha; de esta manera activa la Data Set Ready por su parte, indicando que también el modem llamador está preparado (instante t_3).

5 / El DTE llamador pone en ON la Request to Send para informar al modem que quiere iniciar la transmisión.

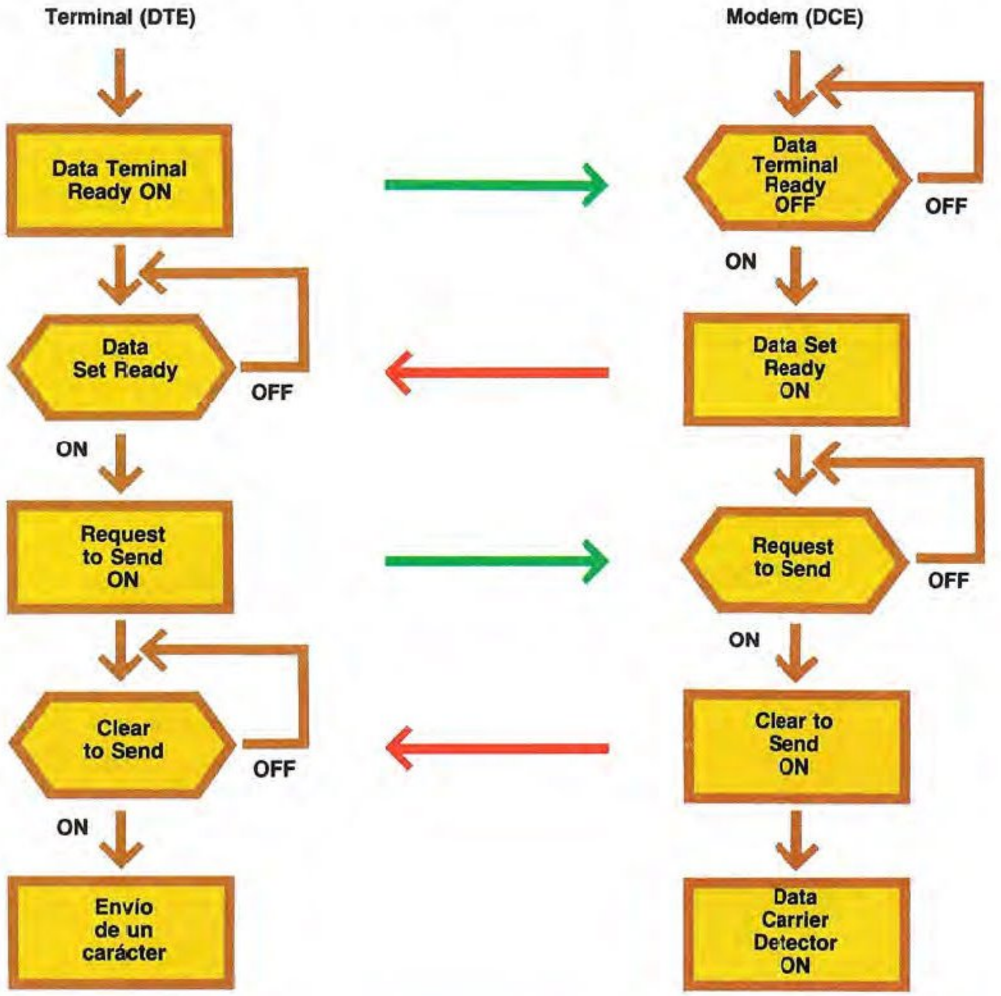
6 / El modem pone en ON la Clear to Send para informar al DTE que todo está preparado para la transmisión (instante t_4).

7 / Ahora empieza la transmisión (instante t_5); la Carrier Detector está en el estado ON para controlar los niveles de las informaciones transmitidas.

8 / Los datos llegan en recepción (t_6); la Carrier Detector está en el estado ON también en la parte del DTE receptor.

En el ejemplo indicado, los intervalos temporales tienen un carácter puramente indicativo, y no tienen en cuenta las características reales de los modems y de las líneas.
En la página siguiente se ha indicado el diagrama de flujo correspondiente al diálogo entre un terminal (DTE) y un modem (DCE) en transmisión; en aquél también se presentan las relaciones causa/efecto entre las distintas fases.

DIALOGO TERMINAL-MODEM EN TRANSMISION



Comunicaciones a cortas distancias

La estandarización de los interfaces de comunicación prescinde de la distancia en la que se produce la transmisión. Sin embargo, es normal encontrarse frente al siguiente problema: un ordenador tiene N interfaces que pueden controlar N dispositivos en la modalidad EIA-RS232C; ¿puede conversar un terminal vídeo en conexión sin modem?

Supongamos que la comunicación a establecer sea de tipo asíncrono.

Si el interfaz del ordenador está diseñado para trabajar sólo a distancias cortas, de él saldrán como norma tres hilos:

- Transmitted Data
- Received Data
- Signal Ground

Si el interfaz del terminal de vídeo trabaja en las mismas condiciones, la conexión que es necesario realizar es muy sencilla; basta tener la precaución de conectar el Transmitted Data del ordenador al Received Data del terminal y viceversa, como se indica en la figura de arriba de la página siguiente.

En estas condiciones, todas las demás señales previstas por la norma EIA no se utilizan, y la conexión es operativa a condición de que los dos dispositivos trabajen con los siguientes parámetros implantados de la siguiente manera:

1 / Número de bits por carácter. Por norma son 7 u 8, según que el bit de paridad esté o no incluido (ver tabla de la pág. 1317)

2 / Número de bits de paro. Normalmente puede escogerse entre 1 o 2 bits

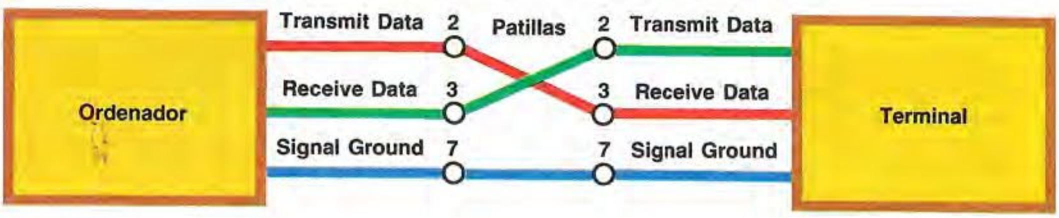
3 / Velocidad de transmisión. Normalmente seleccionable entre 110 y 9600 bps

4 / Control de paridad. Para este control puede elegirse entre NONE (ningún control), EVEN (control de paridad) y ODD (control de no paridad)

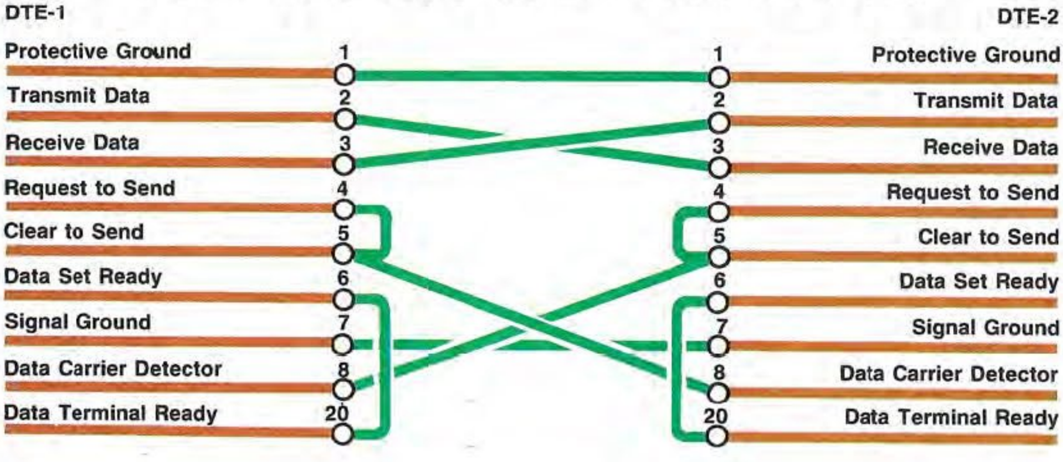
5 / Tipo de flagging o de handshake. Estos conceptos se aclararán después al tratar los protocolos de comunicación. Por ahora es importante observar que también este parámetro debe ser el mismo entre ambos dispositivos: normalmente puede implantarse el tipo XON/XOFF o el tipo ENQ/ACK.

Normalmente, los parámetros citados pueden implantarse tanto en el interfaz del ordenador como en el de los periféricos. Otras veces puede encontrarse ante un periférico «rígido» en la comunicación, en el sentido de que dichos parámetros son fijos. En este caso será necesario adaptar los del ordenador, que es más flexible.
En el caso de que los interfaces de comunicación serie empleen modem, la conexión a distancias cortas podrá establecerse con cables especiales, denominados **modem by-pass**.
En la figura de más abajo de esta página se ha esquematizado una conexión de tipo asíncrono con indicación de las patillas.
Por el lado DTE-1, el Transmitted Data está conectado con el Received Data del lado DTE-2, y viceversa.
La transmisión sólo puede producirse cuando

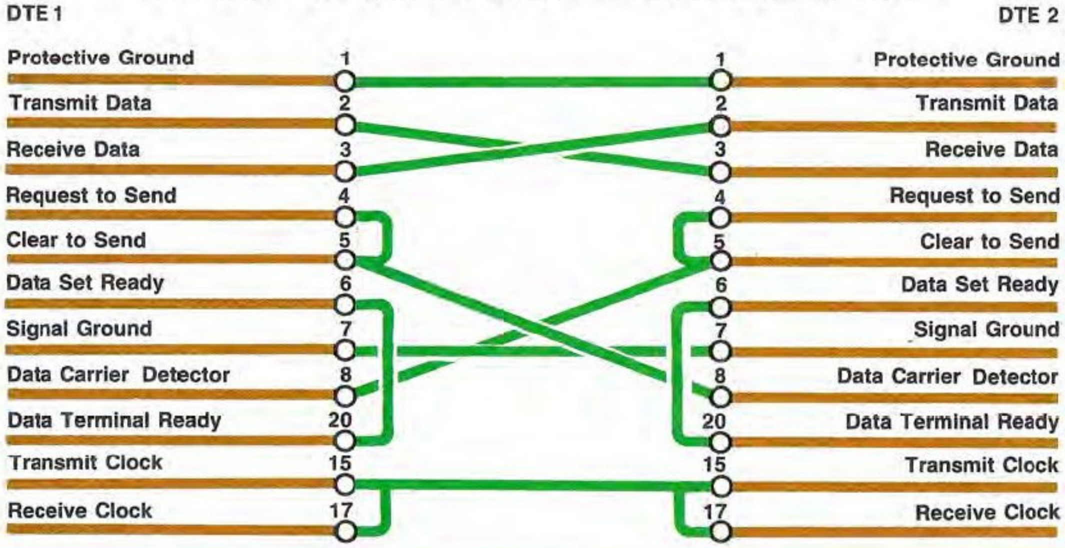
CONEXION DIRECTA A CORTAS DISTANCIAS



CONEXION EN MODALIDAD ASINCRONA A CORTAS DISTANCIAS CON ELIMINACION DEL MODEM



CONEXION EN MODALIDAD SINCRONA A CORTAS DISTANCIAS CON ELIMINACION DEL MODEM



están en ON las señales Request to Send, Clear to Send, Data Set Ready y Data Terminal Ready; las señales Request to Send y Data Terminal Ready son activadas por el DTE (ordenador o terminal), mientras que la Data Set Ready y la Clear to Send son activadas por el modem, que no está. Las conexiones Request to Send/Clear to Send y Data Terminal Ready/Data Set Ready permiten la activación de las cuatro señales necesarias para iniciar la transmisión. Para la recepción debe activarse la Carrier Detector, y esto se obtiene conectando la Request to Send del que transmite a la Carrier Detector del que recibe. La conexión de los Protective Ground puede eliminarse en el caso en que no se desee conectar la masa del DTE-1 a la masa del DTE-2. Arriba se ha representado la conexión a corta distancia en modalidad síncrona. Las conexiones de más con respecto a la figura anterior corresponden a las señales Transmit Clock (o Transmit of Signal Timing) y Receive Clock (o Receive of Signal Timing). La Transmit Clock se emplea en el modem para temporizar el DTE en transmisión, mientras que la Receive Clock se emplea para temporizar el DTE en recepción. Los interfaces síncronos presentes en el ordenador y en el terminal están dotados normalmente de relojes conectados a las patillas 15 y 17. La temporización correcta se asegura conectando entre sí esas patillas y estableciendo la conexión 15-15 entre DTE-1 y DTE-2. Si el interfaz del ordenador y del terminal no están dotados de reloj, es necesaria una temporización exterior, que debe introducir los impulsos en la patilla 15 (o 17) de uno de los dos lados.

Protocolos de comunicación

Las reglas y señales descritas a propósito de los interfaces de comunicación fijan la modalidad a nivel hardware, pero no tienen en cuenta la integridad del mensaje transmitido. Es decir, la unidad receptora no puede establecer qué caracteres se han perdido en la línea, a menos que se adopten reglas ulteriores para la gestión de la comunicación.

Estas reglas existen y se definen como **protocolos**; representan en cierto modo la «gramática» a través de la cual los dispositivos intercambian informaciones. Por tanto, los protocolos de comunicación son los gestores a nivel más elevado de las informaciones y, como los interfaces, están reglamentados por las casas constructoras de ordenadores y por los adecuados Institutos Internacionales para la estandarización. En general, con el término protocolo se entiende una íntima concomitancia entre el hardware y el software: la parte de hardware está representada por los circuitos diseñados para el particular tipo de protocolo que tiene que albergar, mientras que la parte de software está constituida por programas que implantan las reglas que rigen el protocolo.

TEST 22

1 / La comunicación half-duplex permite
- a) la transmisión simultánea de informaciones en ambos sentidos;
- b) la transmisión en ambos sentidos, pero no simultáneamente;
- c) la transmisión en un solo sentido;
- d) la duplicación de informaciones.

2 / La modulación de frecuencia tiene la característica
- a) de no modificar la portadora;
- b) de modificar la amplitud y la frecuencia de la señal;
- c) de modificar sólo la frecuencia de la señal;
- d) de modificar la frecuencia y la fase de la señal.

3 / La configuración multipunto emplea
- a) una sola línea de comunicación;
- b) tantas líneas como esclavos DTE hay;
- c) tantas líneas como fases polling;
- d) tantas líneas como puede gestionar el amo DTE.

4 / La sigla EIA-RS232C indica
- a) los modems asíncronos;
- b) una norma de conexión;
- c) una norma de modulación;
- d) una norma de interfaz de comunicación serie.

5 / La señal Data Set Ready indica
- a) que el modem está prepaprado para transmitir;
- b) que el terminal está prepapado para transmitir;
- c) que en la línea hay portadora;
- d) que el modem ha recibido una petición de transmisión.

6 / Si la Receive Data es activa, también debe activarse
- a) la Signal Ground;
- b) la Transmit Clock;
- c) la Carrier Detector;
- d) la Transmit Data.

7 / ¿Qué señal indica que el DTE está preparado tanto para transmitir como para recibir?
- a) la Clear to Send;
- b) la Data Terminal Ready;
- c) la Request to Send;
- d) la Carrier Detector.

8 / La señal Transmit Clock sirve para
- a) sincronizar el DTE y el DCE;
- b) sincronizar el modem;
- c) sincronizar el DTE en transmisión;
- d) sincronizar los caracteres transmitidos.

Las soluciones, en la pág. 1342

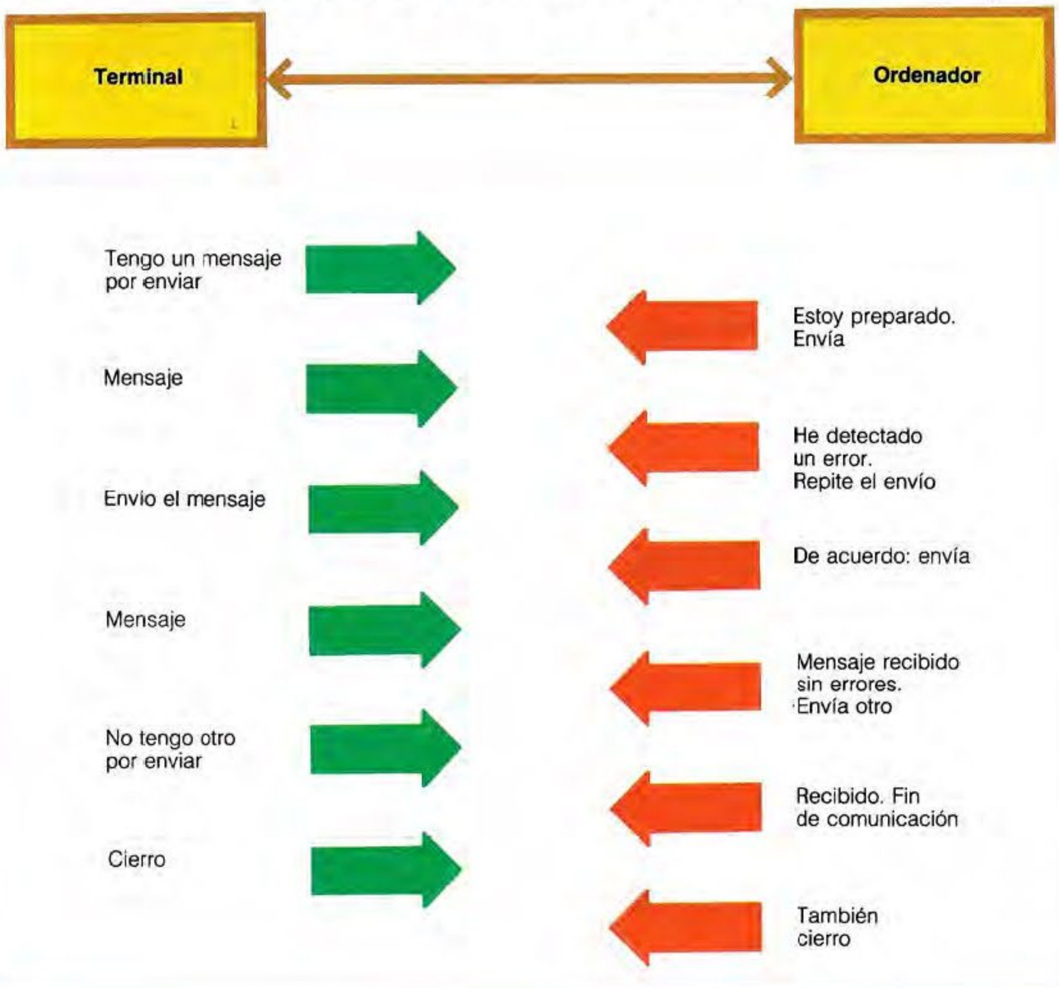
El handshake

Un aspecto importante de los protocolos de comunicación está constituido por el llamado handshake, o flagging, utilizado a efectos de sincronismo. Para no confundirlo con la modalidad de transmisión (síncrona o asíncrona), el handshake tiene por objetivo la integridad de los mensajes. Por ejemplo, si un ordenador debe comunicar con un terminal de vídeo, el handshake garantiza que el terminal consiga visualizar todos los datos enviados al ordenador, sea cual sea la velocidad de transmisión.
Las informaciones intercambiadas en la parte de handshake constituyen un diálogo del tipo:

- Mensaje disponible para ser transmitido
- Señal de espera para aceptar el texto
- Señal de principio de la transmisión del texto
- Aceptación o envío del texto
- Detección de errores en el texto recibido
- Eventual retransmisión del texto
- Fin del texto

Abajo se ha representado un sencillo protocolo. La secuencia muestra un terminal que comunica al ordenador que tiene un mensaje para enviar. El ordenador reconoce el terminal y activa la transmisión. Durante la recepción, el ordenador detecta un error y pide al terminal que retransmita el texto. La segunda vez, el mensaje recibido no contiene errores y el ordenador pide que prosigan las otras comunicaciones. El terminal no tiene ningún otro mensaje para enviar e informa de ello al ordenador. Si el ordenador a su vez tiene un texto para enviar al terminal,

ESQUEMA DE UN PROTOCOLO SENCILLO



puede hacerlo en este momento; como no tiene nada para transmitir, cierra la comunicación y el terminal hace otro tanto.

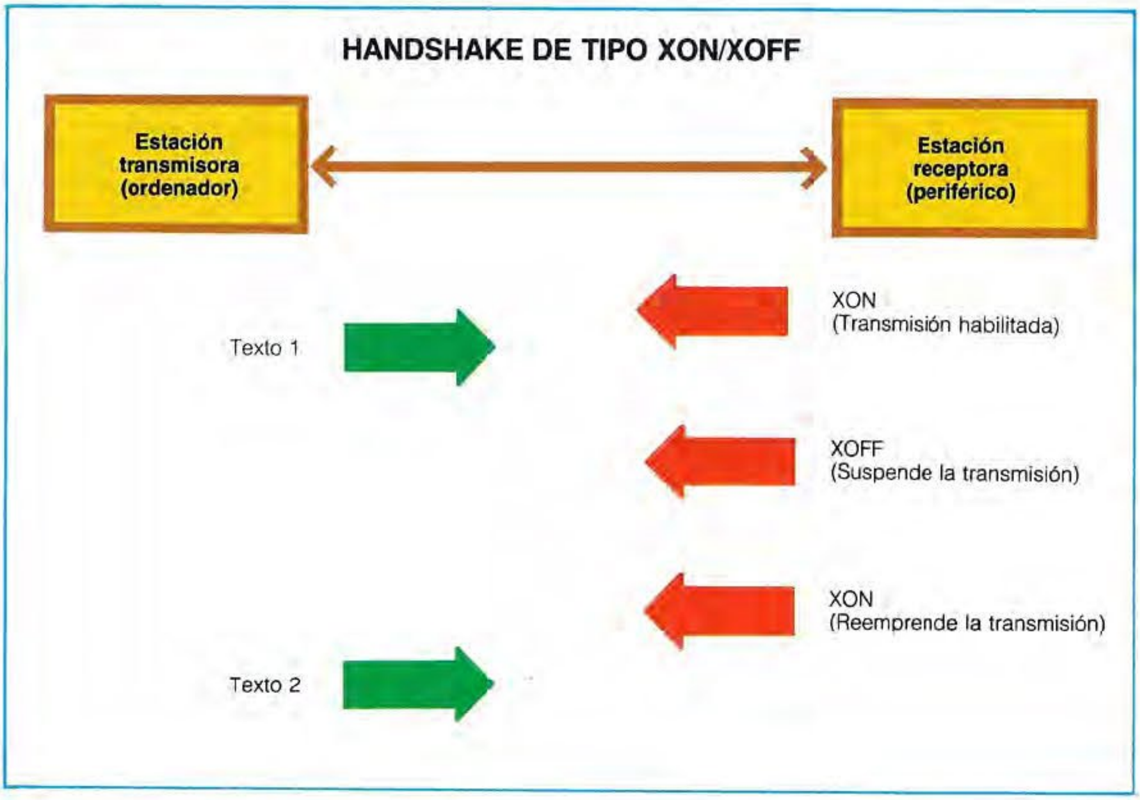
Por las líneas de comunicación gestionadas por protocolos no sólo viajan los datos a transmitir, sino también informaciones adicionales que tienen por objeto garantizar que no se pierda nada. Estas informaciones constituyen la **hoverhead** del protocolo, y al mismo tiempo son un parámetro de juicio del propio protocolo: un protocolo es eficiente si su hoverhead es mínima, porque esto significa que se consigue garantizar la integridad de las informaciones con la mínima superposición de texto adicional y, por tanto, con la mínima pérdida de tiempo.

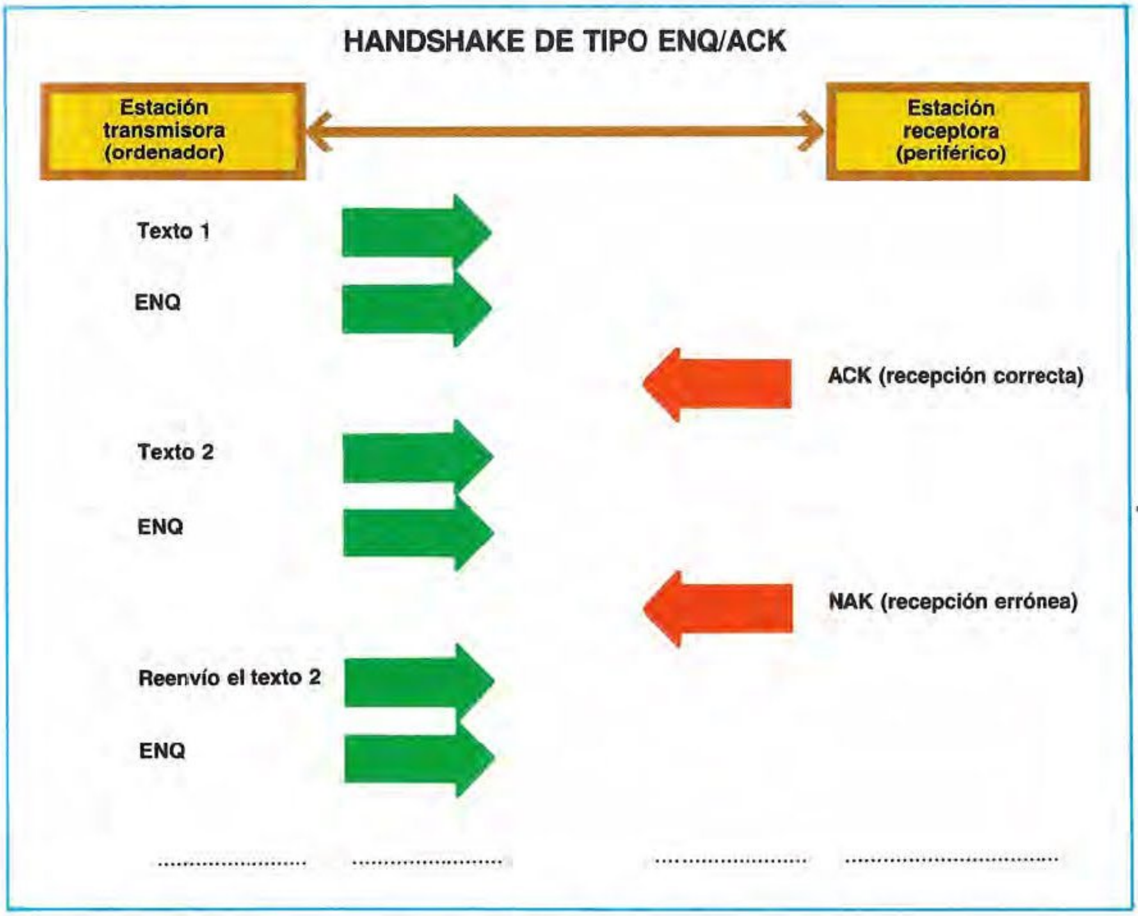
Como existen varios muchos de interacción entre dos estaciones que deben comunicarse, también existen diferentes tipos de hadshake; los principales se describirán más adelante.

Un ejemplo típico que aclara la necesidad de la parte de handshake está constituido por la fase de diálogo de un ordenador con un terminal de vídeo. Supongamos que la conexión se haya efectuado en la modalidad serie asíncrona (EIA-RS232C) a la velocidad de 9600 bps. A esta velocidad, por la línea viajan aproximadamente 960 caracteres en un segundo, y el terminal no conseguiría visualizarlos con la misma velocidad. Por tanto, es necesario que de vez en cuando el periférico informe al ordenador que suspenda la transmisión para tener tiempo de presentar los caracteres recibidos (de hecho, se tiene la misma necesidad para los periféricos como impresoras, unidades de cinta, etc.).

Para limitar este inconveniene se utiliza la «bufferización» de los periféricos. Con este término se indica la capacidad de los diversos dispositivos para almacenar informaciones antes de visualizarlas. Durante la comunicación, los datos transmitidos por el ordenador se acumulan en el buffer interno del terminal, del que después se toman para su proceso. Por tanto, el objeto del buffer es el de hacer de volante o de pulmón durante la comunicación. Cuando la capacidad de almacenamiento del buffer se agota, se debe interrumpir momentáneamente la comunicación para evitar que los datos se pierdan.

El método XON/XOFF. Un método de handshake basado en esta última acción es el llamado **XON/XOFF**, esquematizado en la figura de abajo. El ordenador envía datos que el periférico acumula en su buffer interno. Cuando éste está lleno en sus 2/3 partes, el terminal envía al





ordenador el carácter DC3 (ver la tabla de caracteres ASCII), que tiene el significado de Transmit OFF o XOFF, y el ordenador suspende la transmisión. El terminal presenta los datos acumulados en el buffer y lo vacía; cuando el buffer está lleno en 1/3 de su capacidad, el terminal envía al ordenador el carácter DC1, con el significado de Transmit ON o XON, y el ordenador vuelve a transmitir. Los valores 1/3 y 2/3 no son fijos, sino que varían en función del periférico a gestionar. Algunos periféricos funcionan con llenado y vaciado completo del buffer; los valores utilizados están ligados a razones de optimización de la comunicación en función del hardware. Otros periféricos adoptan el mismo método de handshake que ya no se basa en los caracteres DC1 y DC3, sino en el hecho de que sea alta o baja una determinada señal RS232C, por ejemplo la Clear to Send o la Data Terminal Ready. Esta última forma es la acción más elemental de handshake, y comporta la gestión de señales adicionales al Transmit Data y al Receive Data.

El método ENQ/ACK. Una forma diferente de interacción se basa en el método **Enquire/Acknowledge**, o **ENQ/ACK**, esquematizado en esta página. De esta manera se hace más ágil la petición de retransmisión del texto con la detección de errores por parte del receptor. El que transmite envía el texto seguido del carácter ENQ; el receptor procesa el texto y envía el carácter ACK si todo ha ido bien, o el carácter NAK (Negative Acknowledge) si ha detectado un error. La estación que transmite envía otro texto si ha recibido ACQ, o bien el mismo texto si ha recibido NAK. Con el método de handshake ENQ/ACK se evita el problema de la saturación del buffer, pues los textos transmitidos no superan las longitudes preestablecidas y es siempre el receptor el que proporciona el acuerdo para proseguir.

El aspecto interactivo y de sincronismo sólo es uno de los tantos previstos en la fase de handshake de un protocolo; otras fases, como la detección de errores, se describen en el interior de los propios protocolos.

SOLUCIONES DEL TEST 22

1 / b. La comunicación half-duplex permite la transmisión y la recepción en fases alternas.

2 / c. La modulación de frecuencia funciona modificando la frecuencia de la señal.

3 / a. El amo DTE está conectado a la única línea.

4 / d. La sigla RS232C denota una norma que corresponde al interfaz de comunicación serie.

5 / a. Indica que el modem está preparado para transmitir.

6 / c. La Carrier Detector.

7 / b. La Data Terminal Ready.

8 / c. Sirve para sincronizar el DTE en transmisión.

Teclado del sistema de proceso Commodore Plus 4.



R. Villa/Marka

Juegos de guerra para profesionales

Un campo de fundamental importancia que ha aportado enormes ventajas y desarrollos para la evolución de la electrónica y de los calculadores en particular, es el de la simulación. El concepto que está en la base de la «ciencia» de la simulación es el de reproducir, de la manera más adecuada posible para el usuario, un fenómeno físico o natural, una máquina o un aparato aunque sean complejos, comportamientos y actos humanos, físicos e intelectuales, y en general un conjunto cualquiera de entidades de las descritas anteriormente, simulando la realidad.

Los ejemplos que siguen sirven para ilustrar cómo nace la exigencia de la simulación y cuán importante es, si no indispensable, para la solución de problemáticas complejas.

En lo que respecta al estudio de los fenómenos físicos o naturales, la simulación ha abierto el camino para nuevas hipótesis y teorías proporcionando instrumentos de estudio y la verificación de aquellos modelos.

El ejemplo más clásico de la simulación aplicada a máquinas o a aparatos es el del simulador de vuelos para los pilotos de los aviones. Un simulador de vuelo está constituido por una reproducción exacta del interior de la cabina de pilotaje del avión que se quiere simular, encerrada en un contenedor adecuado montado sobre una plataforma mecánica que puede hacer asumir a la cabina las posiciones que se darían en vuelo durante la maniobra del avión. Todo el conjunto está conectado a un calculador que gestiona los mandos del piloto, los elabora según el modelo previsto, controla los movimientos de la plataforma y controla todos los «instrumentos» de a bordo (en realidad, en los simuladores sólo hay instalados los paneles de mando y control, cuyas teclas, interruptores, llaves, visualizadores, diodos LED, indicadores ópticos y acústicos, están conectados al calculador) para proporcionar al piloto todas las informaciones que tendría en el vuelo real.

Además, existen simuladores que proporcionan al piloto también la visión (diurna o nocturna) que éste tendría al despegar, al volar y al aterrizar en un determinado aeropuerto o al sobrevolar cierto territorio. Las enormes ventajas en términos de seguridad, economía, disponibilidad, versatilidad, etc. de la simulación con respecto a la realidad son evidentes. El adiestramiento de un piloto gracias al uso de un simulador resulta mucho más corto, eficaz y económico, puesto que no depende de la disponibilidad física del verdadero avión (con todos los riesgos, los costes y los tiempos correspondientes a su mantenimiento, asistencia en tierra, asistencia en vuelo, consumo, etc.). En los centros de adiestramiento en tierra, el piloto puede iniciar además su formación y llegar a efectuar despegues y aterrizajes (incluso catastróficos) en cualquier aeropuerto, vuelos sobre cualquier territorio y con cualquier avión, sin elevarse realmente nunca de tierra.

La forma más compleja y evolucionada de simulación es la que va acompañada de fenómenos físicos, aparatos y comportamientos humanos en un conjunto estrechamente relacionado, en el cual cada acción puede dar origen a cambios complejos de todo sistema.

El A.S.T.T. (Action Speed Tactical Trainer, entrenador táctico en tiempo real) pertenece a esta última forma de simulación.

El objeto del A.S.T.T. es permitir a los grupos de mando de la Marina Militar la planificación y la ejecución de ejercicios aeronavales, desde el más sencillo al más complicado, proporcionando a los comandantes y a los oficiales de las diversas unidades todos los instrumentos y los medios necesarios para su adiestramiento táctico (que constituye la forma más elemental de desarrollo de la capacidad de decisión y de mando), tanto en la fase de ejecución como en la de la fase siguiente de playback.

Para poder realizarse en la realidad, un ejercicio aeronaval complejo necesita meses de planificación y preparación, el empleo de centenares de personas (jefes, oficiales y equipo de las diversas unidades, así como todo el personal de tierra) antes, durante y después de su realización, la preparación de las unidades (trabajos en astilleros, pruebas, abastecimientos, aprovisionamiento), la transferencia de las unidades del área de estacionamiento normal a la elegida para el desarrollo de los ejercicios, la preparación del teatro operativo (control de las actividades civiles, como vuelos aéreos de línea o privados, tránsito de embarcaciones de línea, deportivas, pesqueras) y toda una serie de otras actividades colaterales.

Además, un ejercicio real está limitado por la propia realidad. Por ejemplo, en la ejecución de un ejercicio de partes contrapuestas, las varias unidades participantes están asignadas a equi-

pos diferentes, pero claramente estas unidades, como los aparatos y los sistemas de a bordo (sensores, armamentos, etc.), son siempre las de la Marina Militar.

También hay otras limitaciones al uso de las armas, puesto que evidentemente no se podrá abrir fuego real o lanzar misiles contra las propias unidades. Después, existe el problema del análisis global de los resultados del ejercicio, cuyo objeto es el de revisar conjuntamente todas las actividades realizadas por cada unidad para relacionarlas entre sí, con el fin de obtener un cuadro general del cual poder extraer los debidos informes y conclusiones.

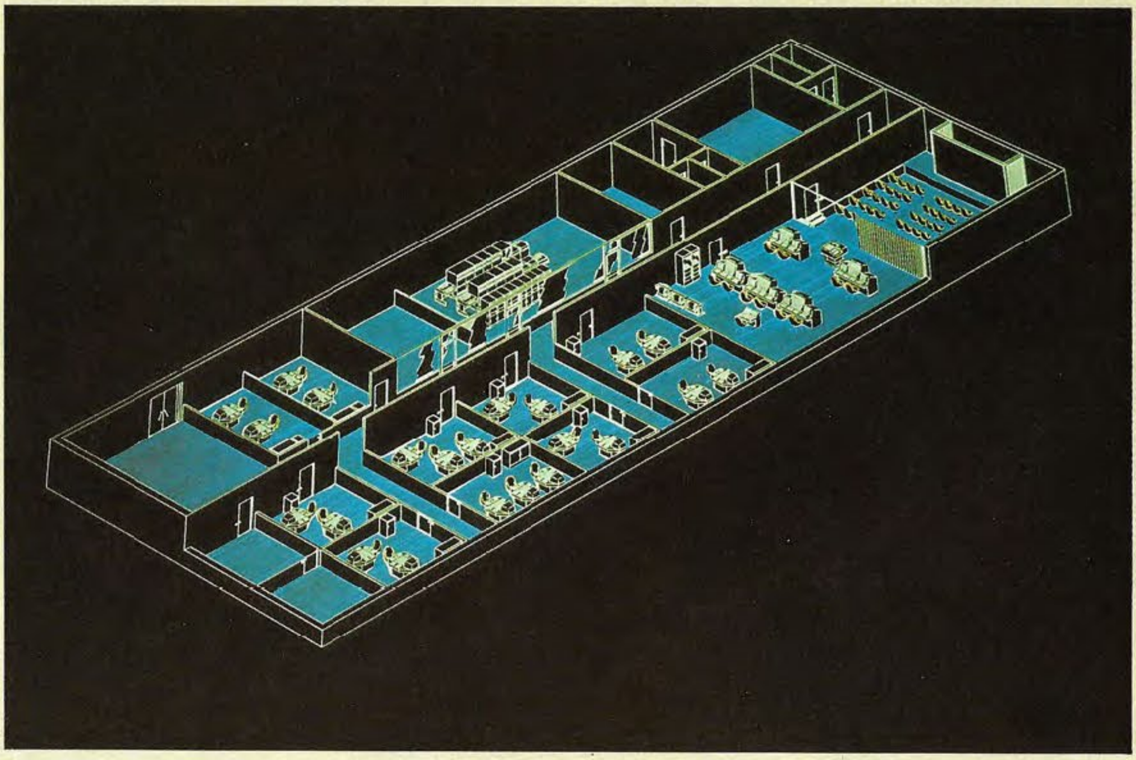
De todo lo dicho resulta claro que un ejercicio real, aun siendo de momento insustituible para verificar el buen funcionamiento de las estructuras, de las unidades y de los aparatos y juzgar el adiestramiento de los hombres, no es ciertamente el mejor método en términos de economía, rapidez, flexibilidad y reproducibilidad para efectuar el adiestramiento táctico de los grupos de mando.

De estos y de otros motivos ha nacido la exigencia de disponer de un sistema de simulación que pueda proporcionar a los jefes y oficiales y a sus colaboradores la posibilidad de adiestrarse sin los inconvenientes, los costos, el tiempo y las limitaciones de los ejercicios reales, conservando al mismo tiempo todas estas ventajas.

El A.S.T.T., realizado por la firma DATAMAT Ingegneria dei Sistemi de Roma y después instalado en el Centro de Addestramento Aeronavale de Tarento, donde es operativo desde enero de 1984, además de satisfacer ampliamente estas exigencias, gracias a sus enormes posibilidades y potencialidades ya descritas, permite el estudio, la ejecución y la verificación de nuevas metodologías tácticas, así como de otras actividades correlacionadas.

El A.S.T.T. está constituido por cinco subsistemas principales repartidos en doce ambientes. Cada uno de los doce cubículos (salas de mando) visible en la figura de esta página puede representar la sala de mando de cualquier tipo de unidad (buque de superficie, avión, helicóptero). A bordo de cada uno de ellos «se embarcan» los oficiales para realizar las misiones utilizando las consolas, los monitores alfanuméricos, los videográficos y los aparatos a su disposición para gestionar y controlar todas las funciones de la unidad.

Disposición de la unidad que compone el sistema A.S.T.T.



Cada grupo de mando trabaja en tiempo real «a bordo» del cubículo como si se encontrase realmente en su correspondiente unidad en un teatro de operaciones verdadero.

La sala de control/auditorio está provista de cinco consolas de dos puestos, mediante las cuales los instructores controlan el curso de los ejercicios y, eventualmente, intervienen a través de los vídeos, los videográficos, los aparatos a su disposición y un large screen projector, *que puede proyectar un tiempo real sobre una pantalla de 3 × 3 metros el curso del ejercicio. Esta sala también se utiliza para efectuar el análisis conclusivo después del ejercicio, durante el cual es posible volver a ver en la pantalla gigante el desarrollo y volver a escuchar todas las eventuales comunicaciones por radio que se han producido y grabadas en tiempo real.*

En la sala de calculadores hay alojados los dos calculadores y todo el hardware adjunto.

El subsistema de cálculo, *que representa el núcleo del sistema, puesto que contiene todo el software que controla el hardware, está compuesto por dos calculadores VAX 11/780 dotados respectivamente de 1,5 Mbyte y 1 Mbyte de memoria central y de 0.5 Mbytes de memoria común, a través de la cual se intercambian los datos, y de 82.5 Mbytes y 55 Mbytes de memoria de masa, más una unidad de cinta digital.*

Cuatro terminales permiten el control de los calculadores y la realización de las operaciones de planificación y preparación de los ejercicios, mientras que una impresora rápida proporciona los datos y los tabulados deseados sobre soporte de papel. También hay un tablero digitalizador, cuyo objeto principal es el de introducir en el calculador los datos correspondientes a los perfiles costeros representados en la carta náutica. La carta náutica de la zona deseada se coloca sobre el tablero y el operador sigue con el cursor el perfil de la línea costera; un adecuado programa adquiere los datos de posición, corrige eventuales errores, efectúa la correlación con las posiciones geográficas verdaderas y archiva en disco el perfil costero así generado.

El subsistema de cálculo se completa después con todo el hardware especial, los generadores para los visualizadores videográficos, los interfaces necesarios que deben conectar todas las consolas, los monitores, los teclados, los videográficos y, en general, todos los aparatos que constituyen el A.S.T.T. distribuidos entre los diversos ambientes.

El subsistema de control está constituido por las consolas de los instructores, por la gran pantalla proyectora y por la unidad de cinta analógica de 8 pistas. Esta última se utiliza para grabar durante el desarrollo del ejercicio hasta 6 transmi-

Sala de calculadores del sistema A.S.T.T.



B. Liotta/Il Dagherrotipo

siones de radio simultáneas, cada una sobre una pista diferente, y además el eventual comentario sobre el ejercicio efectuado por un controlador sobre la séptima pista (la octava pista sirve para la sincronización).

El subsistema de los cubículos *está representado por 10 cubículos, cada uno de los cuales contiene los aparatos necesarios para el adiestramiento de los oficiales. El* subsistema de debriefing *está constituido por el conjunto de las funciones y de los aparatos disponibles para realizar el análisis después del ejercicio (el registro de todos los datos del ejercicio en tiempo real sobre cinta magnética digital, el registro de las comunicaciones de radio y de los comentarios sobre la cinta magnética analógica, la posibilidad de obtener impresiones y tabulados, globales y selectivos, de los datos y de los acontecimientos significativos del ejercicio realizado, el proyector de gran pantalla, etc.).*

El subsistema de las comunicaciones *pone a disposición de las personas que se adiestran un máximo de 18 líneas de comunicación por radio (simuladas como en la realidad con los correspondientes parásitos y limitaciones), tres redes de comunicación RATT (teletipos) y todos los tipos de comunicación de datos y, finalmente, un sistema de comunicación de servicio.*

La realidad que simula el A.S.T.T. es por lo menos compleja y articulada, especialmente en lo que respecta a la correlación y la integración de todas las partes que la componen.

Los elementos fundamentales del escenario simulado son los siguientes.

- *Teatro operativo, constituido por el mar, el cielo y la tierra en las diversas condiciones ambientales posibles (estado del mar, dirección y velocidad de eventuales corrientes marinas, dirección y velocidad del viento, visibilidad, condiciones meteorológicas, propagación de radio y acústica, etc.).*
- *Unidades aeronavales de cualquier tipo, subdivididas en unidades de superficie (desde portaaviones a la más pequeña unidad), submarinos (convencionales y nucleares) y unidades aéreas (desde el caza al helicóptero), con todas sus características estáticas y dinámicas (dimensiones y configuración, superficies equivalentes de radar, sonar e infrarrojos, velocidad, aceleración, alturas o profundidades alcanzables, maniobrabilidad, autonomía, etc.).*
- *Bases aeronavales, centros en tierra o en plataformas marinas para la asistencia y el mando de las unidades aeronavales, con todos los aparatos, sensores sistemas de comunicación, de contramedidas y de armas eventualmente disponibles.*
- *Aparatos y sensores de a bordo, representados por radar, sonar, aparatos de identificación (IFF), medidores de señales electromagnéticas (ESM), sistemas de boyas sonoras, detectores de anomalías magnéticas (MAD), sensores ópticos, etc.*
- *Sistemas de comunicación, subdivididos en comunicación en fonía (radio), transmisiones de datos (data link) y transmisiones vía teletipo (RATT).*
- *Sistemas de contramedidas electrónicas (ECM), constituidos por enmascaradores y perturbadores electromagnéticos, enmascaradores de infrarrojos, etc.*
- *Sistemas de contramedidas acústicas para su empleo en el campo subacuático.*
- *Sistemas de armamento, representados por sistemas de silos, cañones, sistemas de misiles, sistemas antimisiles, sistemas misil/silo, cargas de profundidad, minas, etc., en cuya gestión el software tiene en cuenta las modalidades de empleo real.*
- *Sistemas «humanos», identificados por todas aquellas actividades realizadas por el hombre a bordo de las unidades o en tierra, que, además de constituir una parte integrante de un ejercicio, no están sujetas específicamente al adiestramiento de los equipos de mando. Por ejemplo, las actividades correlacionadas a los sistemas de radar de a bordo de un buque están bajo el control del Mando por lo que respecta a la puesta en marcha, la modalidad de funcionamiento, el apagado de los aparatos, pero son los operadores de radar quienes controlan los aparatos y obtienen de los ecos que hay en las pantallas las correspondientes informaciones. Estas informaciones (típicamente: posición, ruta y velocidad de una determinada unidad), eventualmente correlacionadas entre sí, se presentan después a la sala de mando, que se dispondrán de forma adecuada. En el A.S.T.T., cuyo objeto no es el de adiestrar a los operadores con el radar (para los cuales existen simuladores específicos que pueden reproducir los esquemas de radar con los ecos del caso), todas las*





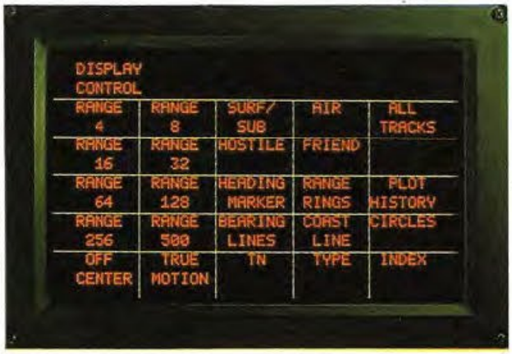


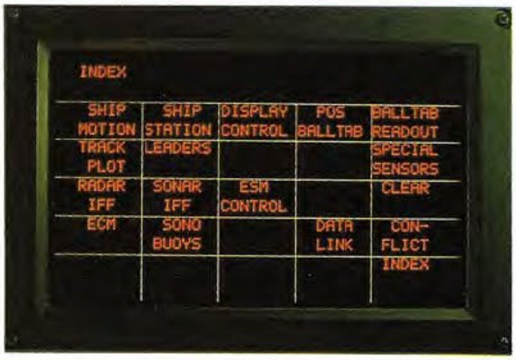
B. Liotta/Il Dagherrotipo

Arriba, las cinco consolas de los instructores y la gran pantalla proyectora. Abajo, dos imágenes de las consolas de cubículo.

operaciones realizadas por los operadores son simuladas por los adecuados programas, que proporcionan después a los equipos de mando las mismas informaciones que recibirían de los radares reales.

Las tres actividades principales del A.S.T.T. son la preparación, la ejecución y el playback de los ejercicios, y a ellas se añaden actividades como la creación y actualización de los archivos de datos correspondientes a las unidades aeronavales, a los escenarios operativos, etc.).
Un grupo de oficiales instructores planifica el ejercicio en función de las exigencias de adiestramiento y del tipo de grupos de mando a ejercitarse (el A.S.T.T. puede realizar simultáneamente hasta un máximo de 16 ejercicios diferentes). Se elige un teatro operativo (dimensiones máximas 1000 × 1000 millas náuticas, de 0 a 900 metros de profundidad y de 0 a 90.000 pies de altitud), cuántas (con un límite máximo de 160) y qué unidades participarán en las acciones y cuál será su disposición inicial.
Estas operaciones se realizan en parte en tablas y en parte a través del coloquio guiado por el calculador que pide al usuario todas las informaciones necesarias, como el nombre del ejercicio, el teatro operativo, las condiciones meteorológicas iniciales del teatro operativo, el estado de la mar, la visibilidad, las condiciones de propagación de radio, el plan de las frecuencias de transmisión a utilizar, las unidades en juego y su disposición inicial.
Todas estas informaciones se memorizan en disco en un archivo del que se reclamarán cuando se ponga en marcha el ejercicio.





Pantallas de las consolas del sistema A.S.T.T.

La ejecución de un ejercicio va precedida de una fase en la que los instructores ilustran a los oficiales a adiestrar respecto a cuáles serán sus misones una vez «subidos a bordo» del cubículo, y comunican además todas las informaciones memorizadas necesarias para el correcto desarrollo del juego de guerra.

Una vez en los cubículos, los oficiales se convierten en los comandantes de sus unidades y desde aquel momento serán sus decisiones las que determinarán el curso del ejercicio.

Desde el interior de un cubículo, los oficiales ven y sienten el mundo exterior (teatro operativo) como si estuviesen realmente en la sala de mando de la unidad simulada. Esto significa, por ejemplo, que si esta unidad se encuentra en el mar con todos los sensores y los sistemas de comunicación apagados y dentro del límite de visibilidad impuesto por el horizonte óptico (eventualmente limitado por las condiciones atmosféricas y otros factores) no hay ninguna otra unidad, en el presentador táctico del cubículo sólo habrá el símbolo de la propia unidad.

Si en el cubículo se activa un radar, y si todas las condiciones necesarias para la descubierta del radar se satisfacen, en el presentador táctico sólo aparecerán los símbolos de las unidades descubiertas por el radar en su correspondiente posición.

En cambio, la situación en el presentador táctico de los instructores indicará, prescindiendo de las indicaciones de los sensores de a bordo de cada unidad sencilla, todas las unidades en su posición real, como si todo el teatro operativo se viese desde arriba sin ninguna limitación.

Durante el desarrollo del juego, los instructores siguen los acontecimientos a través de su consola, el gran proyector de pantalla, los teletipos, los cascos de auriculares, los micrófonos o los altavoces, mandan los llamados fightback targets (unidades sin una sala de mando propiamente dicha, controladas por equipos de mando a través de las mismas consolas de los cubículos, y por otros instructores a través de sus consolas) o las unidades generales, y pueden intervenir modificando en cierta medida los acontecimientos. Por ejemplo, después de haber sostenido un combate con una unidad enemiga, una determinada unidad ha recibido daños que han dejado fuera de servicio el radar y las antenas de los sistemas de comunicación. En este momento, los instructores pueden decidir modificar esta situación para permitir a dicha unidad continuar el ejercicio sin limitaciones.

Los instructores también pueden producir o simular otras averías. Todos estos acontecimientos se memorizan para tenerlos en cuenta durante la fase de análisis.

Cuando el ejercicio ha alcanzado los objetivos prefijados, los instructores detienen la acción y puede procederse a la fase de análisis y de estudio de lo sucedido, que gracias a las posibilidades ofrecidas por el A.S.T.T. resulta muy sencilla, rápida y exenta de errores.

Las posibilidades de volver a ver en la pantalla gigante el desarrollo de todo el ejercicio, de escuchar las comunicaciones por radio, de acelerar el playback, de detenerlo, de posicionarlo directamente en cualquier instante temporal del juego de guerra, así como de disponer de los tabulados de los datos, hacen la fase del análisis después del ejercicio la parte más provechosa del adiestramiento táctico.

Bruno Liotta
Datamat Ingegreria dei Sistemi s.p.a.

El protocolo BSC

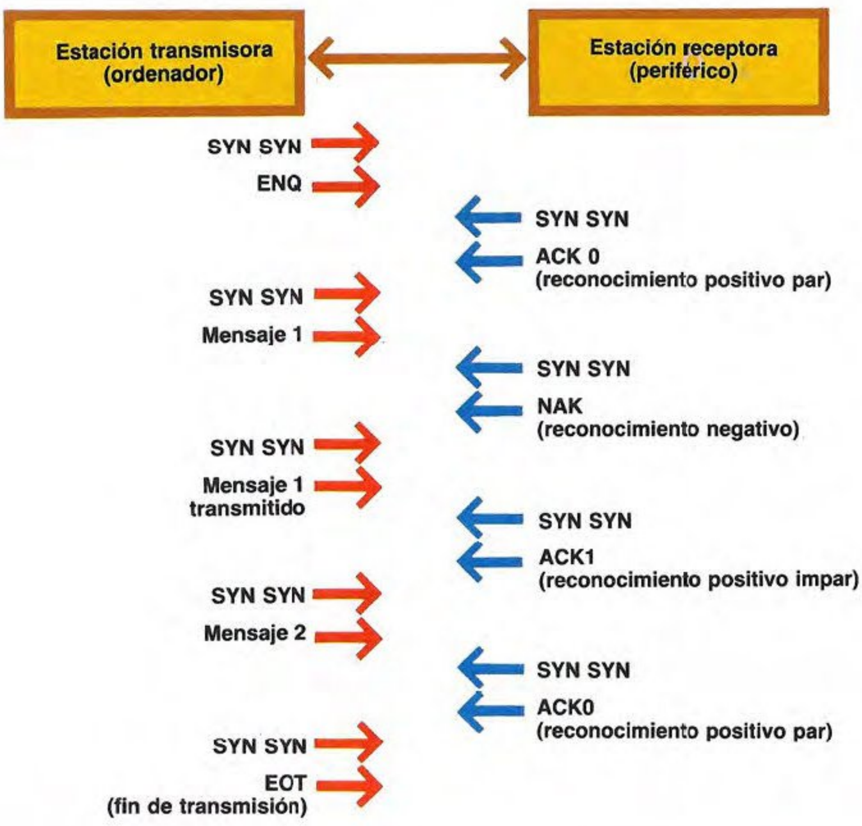
El protocolo BSC (Binary Synchoronous Communication) lo introdujo IBM en 1966. El protocolo cubre el haz de comunicaciones de media y alta velocidad y de tipo half-duplex, en el sentido de que no prevé el intercambio simultáneo de informaciones entre transmisor y receptor. La comunicación es de tipo síncrono y, por tanto, los test van precedidos de un número predeterminado de caracteres SYN (ver la tabla de códigos ASCII y EBCDIC). La secuencia de handshake adoptada, esquematizada abajo, está constituida por una versión modificada del ENQ/ACK. Se han previsto dos caracteres de reconocimiento positivo (ACK0 y ACK1), enviados por el receptor cada vez que el texto está exento de errores. El uso de los ACK pares e impares garantiza la recepción de los reconocimientos y de los mensajes completos. En cambio, si la unidad que transmite no recibe nada después del envío de un texto, transcurrido un determinado tiempo, envía nuevamente el carácter ENQ (ver la figura de arriba de la página siguiente). Si el receptor ha captado el mensaje anterior envía el ACQ alternativo (ACQ1 en la figura citada), y esto significa que sólo se ha perdido el reconocimiento; en cambio, si no ha recibido el mensaje, retransmite el mismo ACK (ACK0 siempre en la misma figura), y el que transmite puede enviar de nuevo el mensaje perdido.

En la figura de abajo de la página siguiente se ha esquematizado el contenido de los bloques de los datos intercambiados con el protocolo BSC. Un bloque de datos suele contener, por lo general, tres elementos: una cabecera (opcional), el texto propiamente dicho y un elemento en la cola para el control de los errores. A estos tres elementos se añaden algunos caracteres especiales:

SYN para establecer la sincronización entre el transmisor y el receptor

SOH Start o Header; indica que lo que sigue es la cabecera del mensaje

HANDSHAKE CON PROTOCOLO BSC



SIGNIFICADO DE LOS ACK ALTERNATIVOS EN EL BSC



FORMATO DE LAS INFORMACIONES TRANSMITIDAS EN BSC



STX Start o Text; indica que lo que sigue es el texto
ETX End o Text; indica que el texto ha acabado.

La cabecera contiene un número fijo de caracteres que acostumbran a identificar a quien transmite y a quien debe recibir y se emplea siempre que en la misma línea de comunicación coexisten más dispositivos. La longitud fija de la cabecera hace inútil la presencia de un carácter de fin de cabecera, mientras que el texto necesita caracteres de principio y de final de texto. El bloque para el control de los errores es generado por el transmisor siguiendo oportunos algoritmos. A medida que se toman los datos, el receptor reconstruye el bloque de control de los errores con el mismo algoritmo, y al final compara el propio bloque reconstruido con el recibido. Si los dos bloques son iguales, el receptor envía la señal ACK (par o impar); si son diferentes, envía la señal NAK para permitir la retransmisión. El bloque para el control de los errores está construido con el método esquematizado en la figura de la página siguiente. Una vez fijado el método de paridad a adoptar (en el caso de la figura citada se adopta el método impar) se realizan dos tipos de controles:

- Vertical (VRC, Vertical Redundancy Check)
- Horizontal (LRC, Longitudinal Redundancy Check)

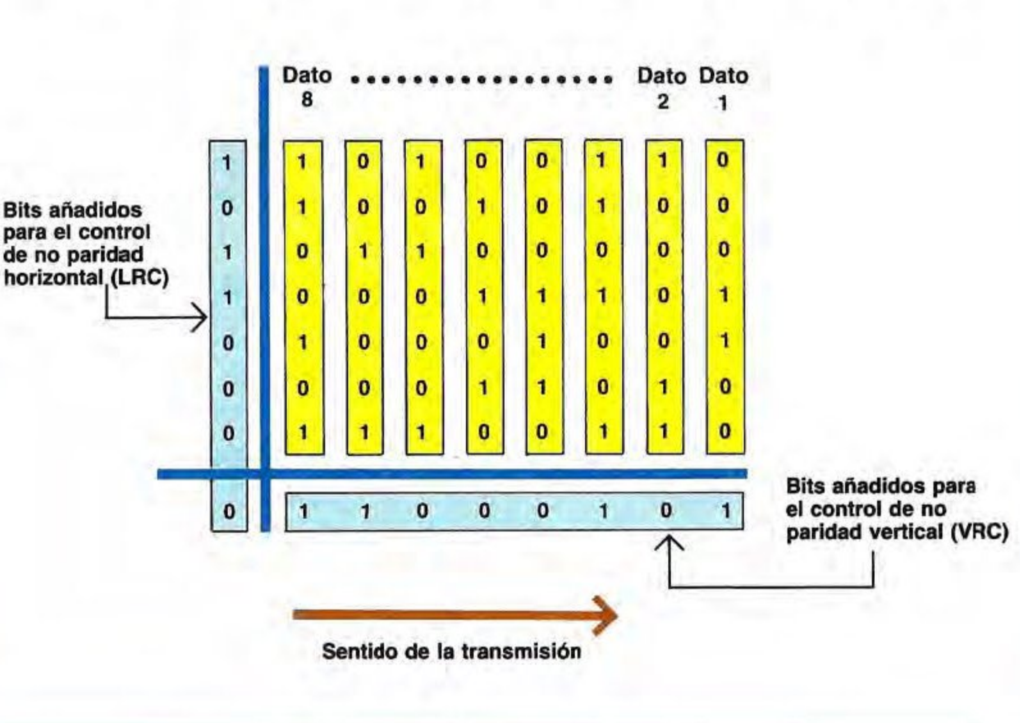
El transmisor añade a cada carácter enviado (representado verticalmente en la figura) un 1 si

el número de bits 1 en el carácter es par o añade un 0 si este número es impar; por tanto, el número completo de bits 1 en cada carácter debe ser impar. Si el receptor descubre que el número de 1 de un carácter es par, detecta un error y corta la transmisión. Este método VRC no es válido cuando el receptor cuenta un número impar de 1 pero el carácter recibido no es igual al transmitido (por una doble inversión, por ejemplo). Por tanto, se añade el control horizontal (LRC) en el que se analizan todos los bits de modo longitudinal y se agrega un 1 si el número de 1 es par o añadiendo un 0 si este número es impar. El control se efectúa para todos los caracteres que constituyen el texto la cabecera (ver la figura de abajo de la página anterior). La columna de bits resultante completada con el bit de no paridad (ver la figura de abajo) es la que se registra en la parte final del control de errores de la figura de abajo de la página anterior. Si la transmisión es correcta, la secuencia de bits registradas por quien transmite debe ser igual a la reconstruida por el receptor. El protocolo BSC, todavía hoy muy usado en las comunicaciones a distancias cortas y largas, es de tipo half-duplex, y por tanto no permite el intercambio simultáneo de informaciones en ambos sentidos. Esto significa que en los momentos de tráfico elevado, en las líneas se crean largas esperas, que podrían reducirse sensiblemente si el coloquio fuese del tipo full-duplex.

El protocolo HDLC

Este protocolo (High level Data Link Control) fue desarrollado por la ISO (International Standard Organization) y posee una arquitectura que puede albergar comunicaciones half-duplex y full-duplex.
Desde el punto de vista de control del tráfico, el protocolo no hace referencia a unidades transmisoras y receptoras, sino a estaciones primarias y estaciones secundarias: la estación primaria es la que inicia el diálogo, y las estaciones secundarias son las que responden. Esta esquematización está ligada al tipo de configuración que se quiera adoptar (punto a punto, multipunto, etc.), y no significa necesariamente que la estación secundaria pueda hablar sólo cuando es interrogada. En general, el diálogo entre las estaciones puede ser asíncrono, en el senti-

CONTROL DE NO PARIDAD HORIZONTAL (LRC) Y VERTICAL (VRC)



do de que cada una puede comunicar cuando tiene necesidad.

El formato de las informaciones transmitidas en HDLC se ha esquematizado abajo. El bloque completo se define como **frame**, y además el mensaje contiene informaciones de control. El mensaje también puede estar ausente del frame y, en este caso, las informaciones de control sirven para direccionar o para poner en estados conocidos las estaciones primarias y secundarias. Todos los frames están numerados en secuencia, y cada uno de ellos contiene el contado de los frames que una estación ha transmitido a otra, así como el número de frames que quien transmite espera recibir a su vez de la estación receptora. Por tanto, la parte de handshake es interior al propio protocolo.

Los principales campos que aparecen en un frame son los siguientes:

Flag de apertura Está compuesto por una secuencia de bits única en todo el frame y de longitud fija (por ejemplo la secuencia 01111110). El flag de apertura informa al que recibe que está empezando un frame. Las estaciones secundarias se ponen en espera de recibir el flag de apertura después de una dirección de estación.

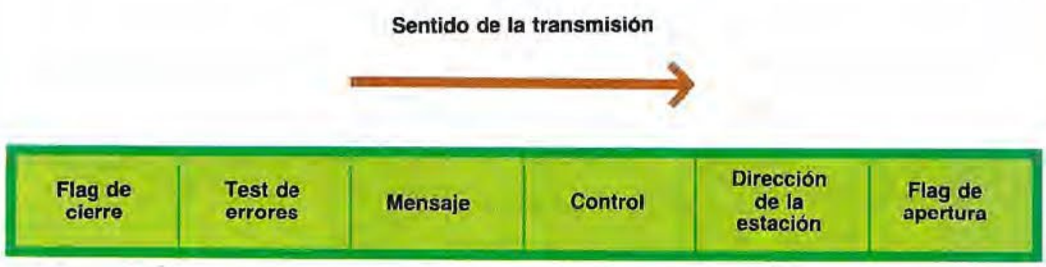
Dirección de estación Identifica el destinatario del mensaje.

Control Permite la eliminación de los frames duplicados, omitidos o erróneos. Contiene, entre otros, dos tipos de información: el número de frames transmitidos (NS) y el número de frames recibidos (NR). Los valores de NS y NR son diferentes para cada estación. Con referencia al diálogo entre dos estaciones, éstos tienen el siguiente significado: «el número del frame que está enviando ahora a esta estación es NS; el número del próximo frame que espero recibir de esta estación es NR». Cuando un frame es recibido por una estación, ésta compara su propio NR con el valor del NS contenido en el frame; si los dos números son iguales, y si no hay errores, el frame se acepta y los contados de NR y NS se actualizan.

Además de NR y NS, el bloque de control contiene informaciones adicionales que permiten transmitir muchos frames antes de que el receptor dé la respuesta de protocolo.

Mensaje Puede ser de cualquier naturaleza (EBCDIC, ASCII, binario u otro) y tener una

FORMATO DE LAS INFORMACIONES TRANSMITIDAS EN HDLC



longitud cualquiera. Esto se obtiene gracias a la combinación de informaciones contenidas en la dirección de la estación y en el bloque de control: si una estación sólo puede hablar en ASCII, a su dirección hay preasociado el código ASCII. Un nivel de seguridad adicional se obtiene especificando en el bloque de control que el mensaje que sigue está escrito en ASCII.

Test errores

Está construido con métodos escritos en el protocolo BSC, o bien empleando métodos similares. El transmisor envía el propio bloque para el test de los errores y el receptor compara éste con el calculado que tiene en el frame; si los dos bloques no son iguales, el receptor rechaza el frame y a la primera oportunidad cerrará su transmisión. Como puede observarse, no es necesario interrumpir la comunicación para retransmitir mensajes; la petición de la transmisión (handshake) es interna al propio protocolo, y por esta razón el protocolo HDLC puede definirse de tipo full-duplex. Abajo se ha esquematizado la filosofía de handshake implantada en el protocolo.

Flag de cierre

Tiene el mismo formato del flag de apertura e identifica el final del frame transmitido.

El protocolo HDLC constituye hoy en día la base para la realización de redes de ordenador bastante complejas y se emplea para controlar periféricos rápidos (unidades de disco y cinta, instrumentos electrónicos de medida) a cortas distancias (pocos metros). El protocolo es el tipo «byte serie, bit paralelo» y permite conversar a un número máximo de 15 instrumentos a alta velocidad (hasta un millón de caracteres por segundo).
Además de definir las reglas de comunicación, el protocolo IEEE-488 también fija una norma a nivel de interfaz, estableciendo reglas precisas correspondientes a los niveles de tensión, a los conectores y a los cables de conexión. Un cable IEEE-488 está constituido por 16 hilos, 8 de los cuales se emplean para la transmisión de los datos y los otros 8 para el control de los datos transmitidos. Por este motivo se dice que el protocolo es paralelo: un carácter se presenta al receptor de manera simultánea sobre los ocho hilos adecuados para la transmisión de los datos. La conexión de los dispositivos a la línea de comunicación (definida como bus en el protocolo) se ha esquematizado abajo. Cada dispositivo está identificado por un número de reconoci-

FORMATO DE LAS INFORMACIONES TRANSMITIDAS EN HDLC



miento (dirección), variable entre 1 y 15, y cada uno de ellos puede enviar o recibir. Sin embargo, el protocolo establece que en un determinado instante en el bus sólo haya un transmisor y uno o más receptores: el bloque de datos que el transmisor envía puede ser recibido por el receptor (o por los receptores) cuya dirección está incluida en el bloque transmitido. Un instante después puede cambiar el transmisor, así como pueden cambiar los receptores.

En la página siguiente se ha detallado la conexión paralela de dos dispositivos con la relación de las líneas empleadas.

Las **líneas de datos** están compuestas por 8 hilos por los que transitan caracteres ASCII en paralelo; 7 hilos se utilizan para la información y el octavo para el control de paridad. Las líneas también conducen los comandos que quieren intercambiar los dispositivos y la dirección de los seleccionados para la escucha.

Las **líneas de handshake** son tres y pueden emplearse para coordinar la transferencia de informaciones por el bus. La comunicación es asíncrona y la velocidad de transferencia entre el dispositivo emisor y uno o dos dispositivos receptores se adecua automáticamente a la velocidad del dispositivo más lento.

Las diversas líneas de handshake tienen el siguiente significado:

DAV — Data Available, controlada por el que transmite para informar al que recibe que hay datos en el bus

NRFD — Not Ready For Data, controlada por quien recibe para informar al que transmite que uno de los receptores todavía no está preparado

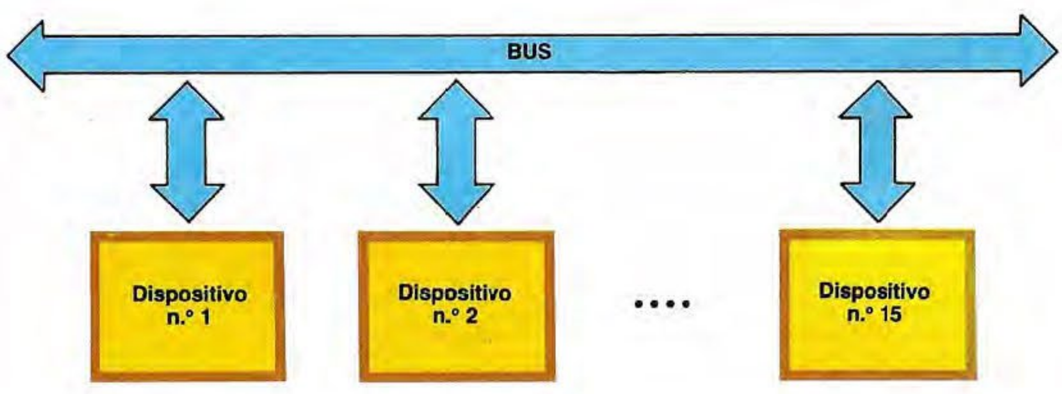
NDAC — Not Data Accepted, gestionada por el receptor para informar que todos los datos se han recibido y, por tanto, no pueden aceptarse otros a menos que se inicie un nuevo ciclo. Esta línea es levantada por el dispositivo de escucha más lento al final de una transmisión.

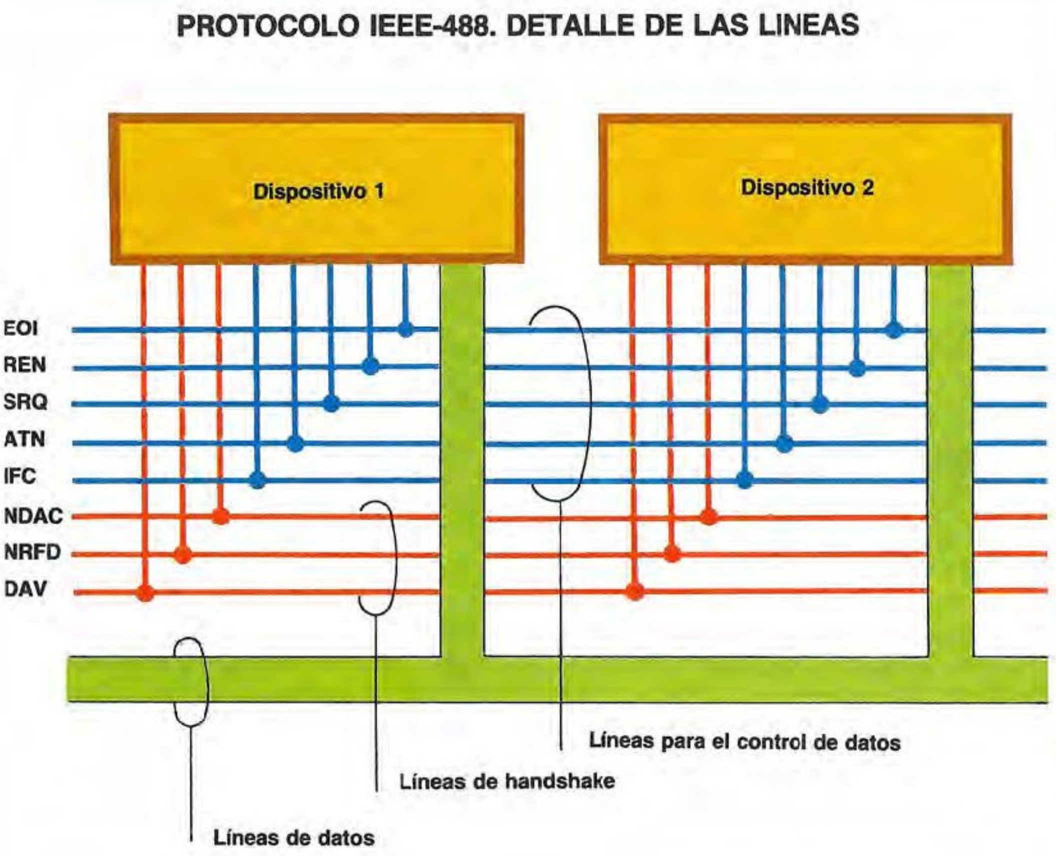
Las **líneas de control de datos** son para ordenar el flujo de informaciones en el bus. El significado de cada una de ellas es el siguiente:

IFC — Interface Clear, se utiliza para efectuar un reset sobre el bus, poniéndolo en un estado de «partida»

ATN — Attention, obliga a todos los dispositivos a interpretar el contenido del bus como «comando» (línea alta) o como «dato» (línea baja)
Decir que en el bus hay «comandos» significa que en esta fase se está estableciendo que debe transmitirse o que debe recibirse; la dirección del emisor o de los receptores se registra en la línea de datos. Decir que en el bus hay «datos» significa que los dispositivos habilitados pueden intercambiarse informaciones

CONEXION DE DISPOSITIVOS EN EL PROTOCOLO IEEE-488





SRQ Service Request, es la línea que permite el diálogo entre dispositivos bajo interrupción (interrupt). Cuando se emplea esta línea, uno de los 15 instrumentos conectados al bus asume las funciones de controlador y establece cómo debe proseguir el diálogo. La situación que puede imaginarse es la siguiente: hay un diálogo entre emisor y receptor; la línea SRQ para a ser alta y eso significa que otro instrumento necesita atención. El controlador interroga a los diversos dispositivos (polling) y apenas descubre el que ha levantado la línea, le da la posibilidad de transmitir

REN Remote Enable, está gestionada por los instrumentos que pueden encontrarse en dos estados: local y remoto. Cuando están en local pueden aceptar comandos de un operador que actúa, por ejemplo, sobre un teclado conectado al instrumento; cuando están en remoto pueden aceptar comandos y datos del bus

EOI End Or Identify, indica el final de un grupo de informaciones transmitido.

El protocolo IEEE-488 es uno de los más eficientes en términos de velocidad de comunicación y flexibilidad de uso. Su empleo está muy difundido en el control de los instrumentos electrónicos y actualmente empieza a difundirse también en el empleo sobre periféricos tradicionales y rápidos. No se utiliza en las comunicaciones a distancia y, por tanto, no es adecuado para las redes de ordenadores bastante dispersas.

Redes para comunicación de datos

Todos los elementos utilizados para la teletransmisión de datos, como interfaces, líneas, modalidad de transmisión y protocolos, encuentran su síntesis en las redes de comunicación.

Una red está compuesta en general por un elemento hardware y por un elemento software; el primero está constituido por los aparatos que permiten las comunicaciones, y el segundo organiza y gestiona la red. La característica de base de una red es el empleo «compartido» de sus recursos. El tèrmino recurso comprende cualquier cosa que la red pueda ofrecer a los usuarios: una impresora, una cinta magnética, un programa o un banco de datos, residentes localmente o a distancia.
La estructura de una red, en general, puede ser de dos tipos: **concentrada** o **distribuida**.
En la estructura concentrada hay la presencia de un mainframe que realiza el grueso de los procesos necesarios desde el punto de vista del acceso a la red. A este punto están enlazacos terminales de vídeo o dispositivos de entrada-/salida, con la misión de interrogar el mainframe. Por tanto, la red concentrada se basa en la centralización de las informaciones. En cambio, la red distribuida está constituida por varias unidades de proceso que tienen más o menos la misma potencia; cada unidad está definida como **nodo**, y puede ser un mainframe o bien (más frecuentemente) un miniordenador. A cada nodo hay conectado un cierto número de periféricos, conectados localmente o a distancia. Por su misma composición, la red de estructura distribuida potencia y facilita la difusón de informaciones sobre áreas geográficas bastante grandes. Este último tipo de red actualmente está muy difundido, puesto que elimina el «cuello de botella» del ordenador central, que en algunos casos no puede hacer frente a grandes peticiones de proceso. Más ordenadores conectados entre sí con las modalidades descritas más adelante pueden dividirse el trabajo, y si uno de ellos falla, los otros asumen su actividad.
La creación de redes de calculadores permite hoy en día el acceso a grandes bancos de datos. Este acceso es posible también a nivel doméstico: basta dotar al televisor de casa con un particular dispositivo para realizar una conexión con el nodo más cercano, y una red de ordenador puede proporcionar las informaciones más diversas: horarios de trenes y aviones, farmacias abiertas, noticiarios, etc. En algunos casos es posible conectar con la red telefónica normal a través de la cual puede efectuarse directamente una reserva de viaje, enviar comunicaciones a otros usuarios, o pedir al banco datos o noticias de naturaleza varia.

Topología y características de las redes

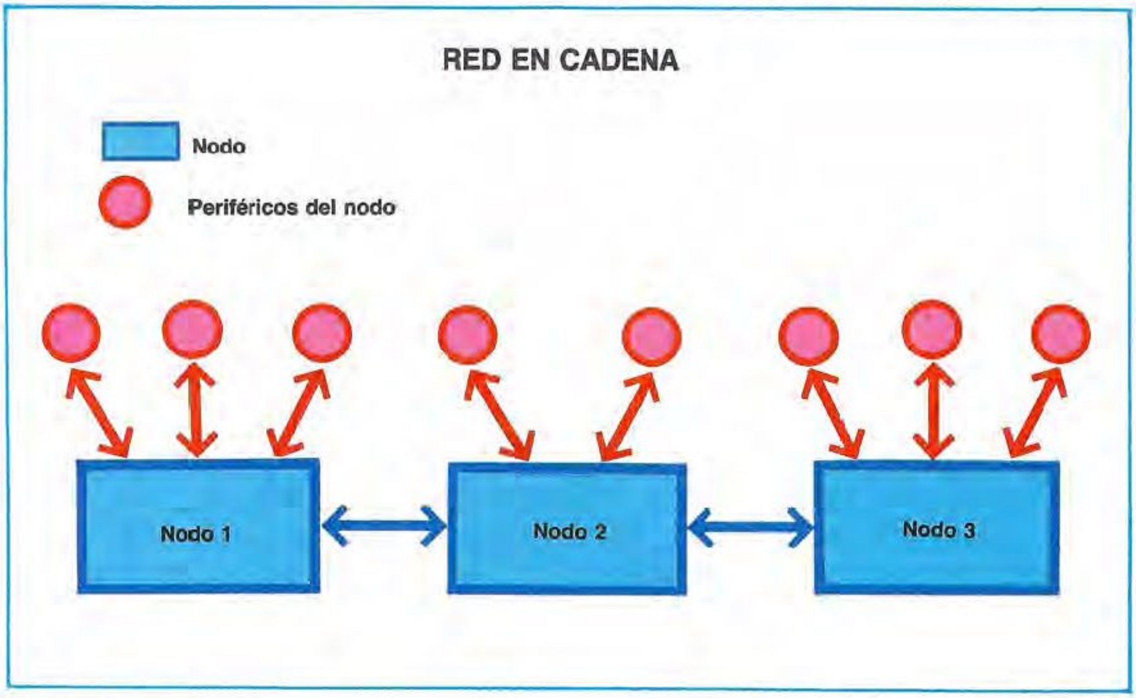
Los nodos de una red pueden conectarse con varias modalidades, las cuales entran en el ámbito de la topología descrita a continuación.

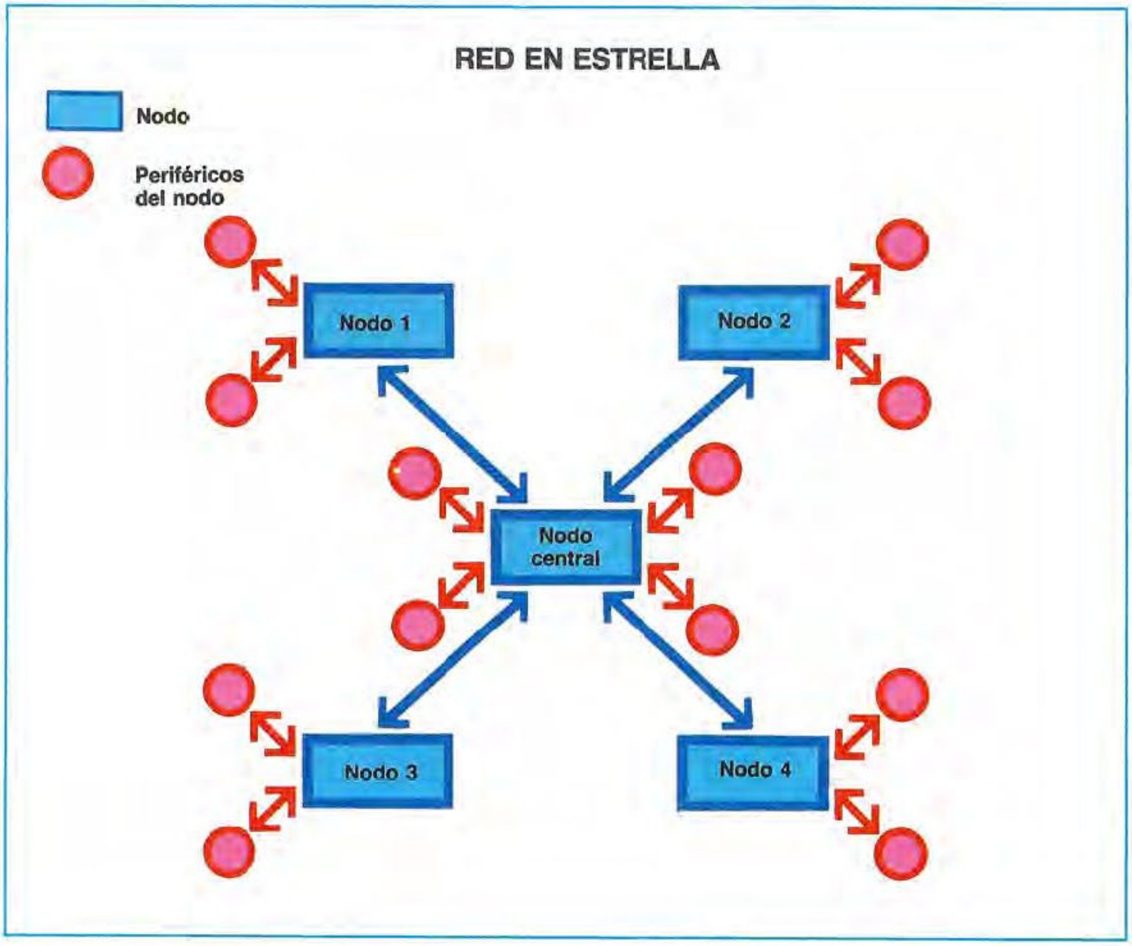
Red en cadena. Es la generalización de las conexiones punto a punto (ver la figura de arriba de la página siguiente); los nodos están conectados en cascada y las comunicaciones que parten del nodo 1 destinadas al nodo 3 deben pasar por el nodo 2. La red es sencilla de construir pero tiene la desventaja de depender del nodo 2 para las comunicaciones entre los nodos extremos. Si el nodo 2 se avería, las comunicaciones quedan interrumpidas. La topología en cadena se utiliza cuando los nodos simples sienen misiones bien precisas y no es vital la comunicación entre todos los nodos de la red.

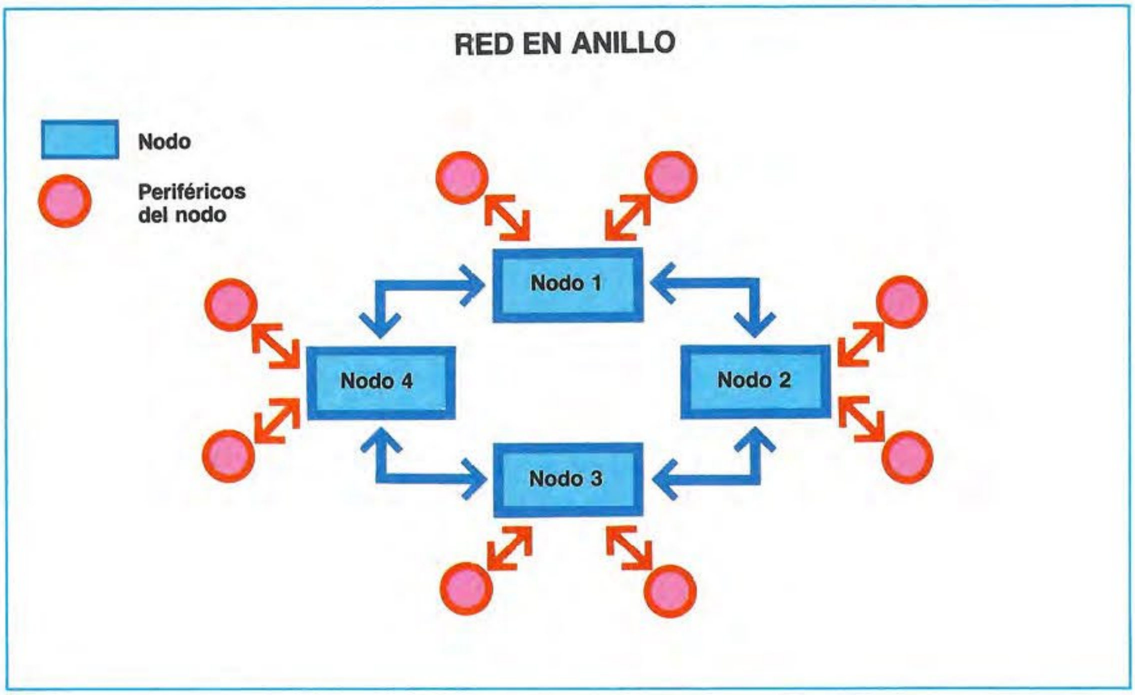
Red en estrella. Es la extensión de una red concentrada (ver la figura de abajo de la página siguiente). El nodo central tiene la misión de distribuir el grueso de las informaciones a los nodos periféricos, que ponen a disposición del mundo exterior la parte de datos a los que pueden acceder. Por tanto, la comunicación entre los nodos periféricos debe transitar a través del nodo central al que está supeditada toda la red.

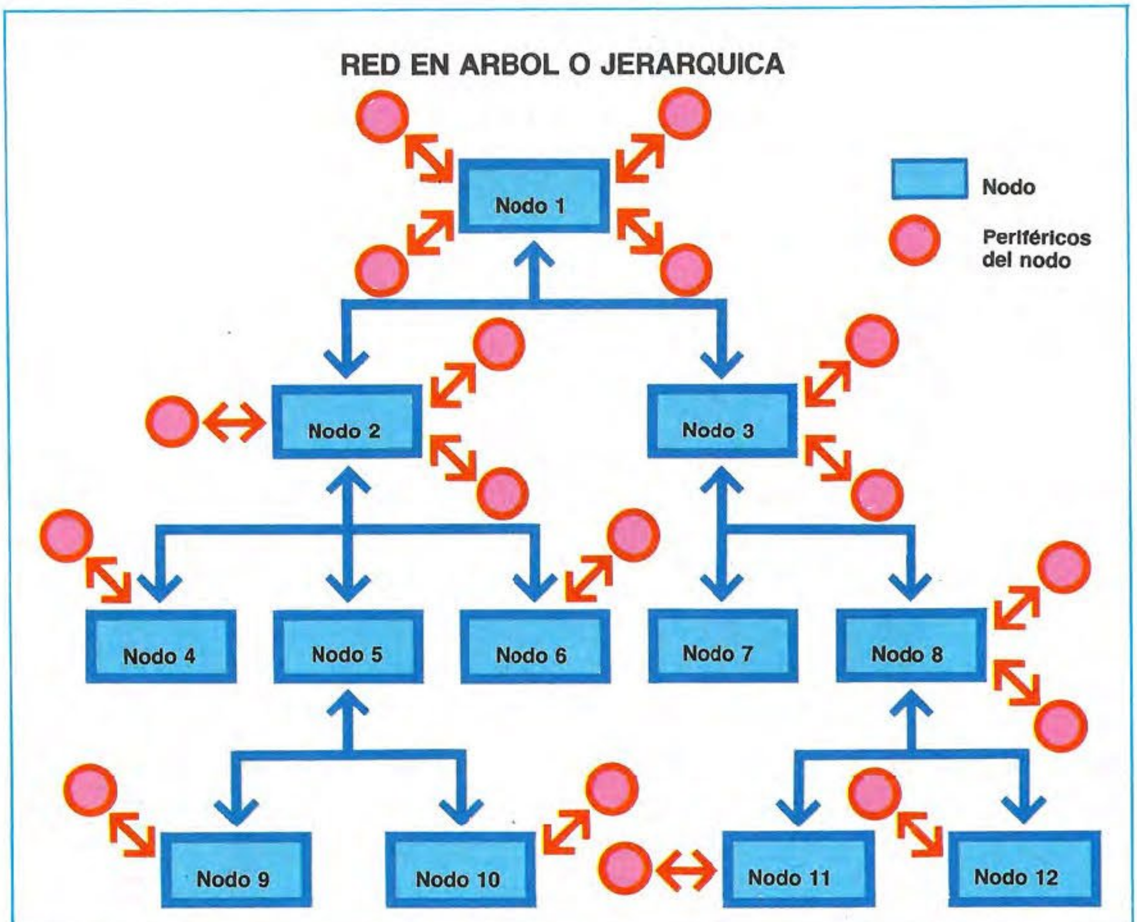
Red en anillo. Es el primer ejemplo de red no supeditada a un nodo (ver la figura de arriba de la página 1358). Las informaciones que parten del nodo 1 y se dirigen al nodo 3 pueden transitar indiferentemente a través del nodo 2 o a través del nodo 4. Normalmente se establece una vía de comunicación habitual (por ejemplo a través del nodo 2), pero si éste se avería se utiliza la línea alternativa (a través del nodo 4). Por tanto, en la red en anillo se tiene la ventaja de poder comunicar también cuando se avería un nodo; la única desventaja reside en el costo, puesto que de cada nodo deben partir siempre dos líneas de comunicación. En la red en anillo no hay ningún ordenador (o nodo) privilegiado con respecto a los demás, y normalmente todos tienen la misma potencia de proceso.

Red de árbol o jerárquica. Se utiliza para que los nodos realicen simples funciones bien precisas e identificadas por una relación de dependencia en las comparaciones de los nodos de jerarquía más alta (ver la figura de abajo de la pág. 1358). Por ejemplo, el nodo 2 debe transfe-


RED EN CADENA
Nodo
Periféricos del nodo
Nodo 1
Nodo 2
Nodo 3


RED EN ESTRELLA
Nodo
Periféricos del nodo
Nodo 1
Nodo 2
Nodo central
Nodo 3
Nodo 4


RED EN ANILLO
Nodo
Periféricos del nodo
Nodo 1
Nodo 2
Nodo 3
Nodo 4

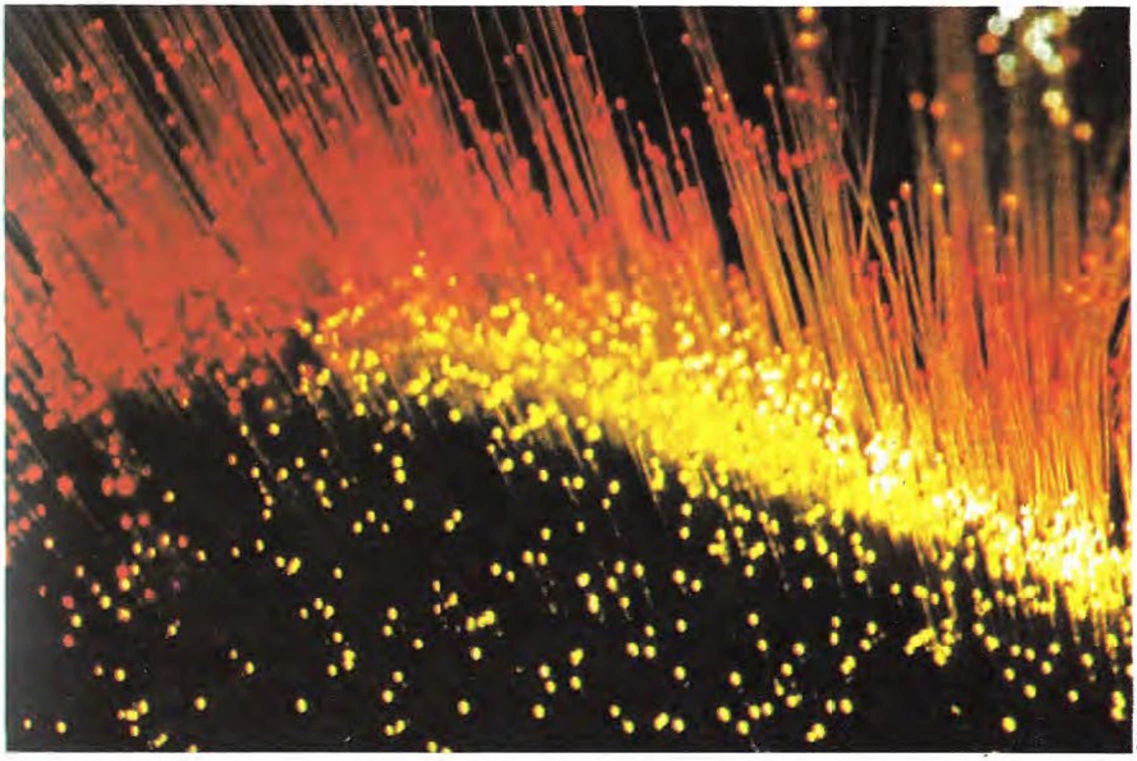

RED EN ARBOL O JERARQUICA
Nodo
Periféricos del nodo
Nodo 1
Nodo 2
Nodo 3
Nodo 4
Nodo 5
Nodo 6
Nodo 7
Nodo 8
Nodo 9
Nodo 10
Nodo 11
Nodo 12

rir informaciones a los nodos 4, 5 y 6 y el nodo 5 es responsable de la transferencia de datos a los nodos 9 y 10. Si del nodo 10 se quiere comunicar con el nodo 6, el nodo 10 pasa la petición al nodo de grado jerárquico más alto (el nodo 5), éste pasa la petición al nodo 2, y de éste, la comunicación llega al nodo 6. El mismo concepto vale también para los procesos que pueden efectuar los nodos simples; si una petición de proceso no pertenece al nodo 9 al que está sometida, se «girará» al nodo que hay arriba, y así sucesivamente. La topología jerárquica está difundida en las redes donde es fundamental la seguridad del acceso a las informaciones y se utiliza en las redes privadas que conectan agencias de bancos en aplicaciones similares. Las topologías acabadas de describir se mezclan para dar lugar a redes más complejas cuya estructura goza de las propiedades de cada topología simple constituyente. En las redes complejas tienen gran importancia dos propiedades definidas como **store and forward** y **packet switching**.

El store and forward permite la transmisión de mensajes entre dos nodos que no tienen una conexión directa. En una red compleja cualquiera, un mensaje se origina en un nodo y tiene un nodo destinatario. En los modernos protocolos de comunicación (ver HDLC), el nombre de quién transmite y la dirección del receptor están escritos en el propio mensaje. Por tanto, el mensaje deberá circular a través de nodos que no son los destinatarios, y éstos tienen una importante función en la red: deben memorizar (store) el mensaje si la línea en la que éste debe viajar está ocupada; y viceversa, deben hacer proseguir (forward) el mensaje por la línea si ésta está libre. Por ejemplo, en la figura de arriba de la pág. 1357, ésta es la función del nodo 2 cuando los nodos 1 y 3 quieren comunicar. El packet switching es el método más nuevo y rápido con que puede transitar un mensaje a través de nodos no destinatarios. Con él, las informaciones viajan entre dos nodos siguiendo una vía no determinada a priori. Es decir, si existen varias vías que conectan dos nodos, el método es capaz de determinar cuál es la vía libre en un determinado momento. Con el packet switching, las informaciones se empaquetan para ser transmitidas siempre con una longitud fija, y esto confiere velocidad a la comunicación.

Piénsese como ejemplo en un autobús con cincuenta personas a bordo que debe atravesar una ciudad; el conductor cambiará de calle si

Conducción de luz polícroma en un haz de fibras ópticas.



Schott

descubre que la que sigue está colapsada. Las cincuenta personas son el sinónimo de autobús completamente lleno para hacer que el viaje sea rentable; lo mismo sucede para las informaciones empaquetadas y de longitud fija.

Redes públicas

La difusión de los mini y microordenadores, así como de los ordenadores personales, ha obligado a las compañías telefónicas a crear sistemas para el intercambio de informaciones.

El concepto nuevo, respecto a las telecomunicaciones tradicionales, que consiste en el uso de las líneas telefónicas normales, reside en dos factores: la existencia de ordenadores (de propiedad de las compañías telefónicas) que tienen la misión de gestionar el tráfico de informaciones por la red, y un método diferente de contabilizar el tiempo de comunicación.

Antes de la aparición de las redes públicas (definidas precisamente con las siglas PDN, Public Data Network), dos o más usuarios, para comunicarse a distancia, debían solicitar una o más líneas telefónicas dedicadas a este objeto, y ser autorizados. Podían utilizarse redes conmutadas, en las que se obtenía línea marcando solamente el número asignado, pero a este tipo de conexión correspondían velocidades de comunicación bajas (de sólo 300 bps) y, en cada caso, si en la línea abierta había silencios en la comunicación, el tiempo también se contabilizaba. En las redes públicas, la contabilización está ligada sólo al volumen de las informaciones transmitidas, por lo que los silencios de transmisión no gravan sobre los costos. Este resultado está ligado al uso de packet switching y de los modernos protocolos, gracias a los cuales, el camino de un mensaje por la red no es siempre el mismo. Quien no transmite es ignorado por la red, mientras quien transmite tiene la garantía de la red de que el mensaje llegará a su destino, no importa por que vía. Ya existen redes con estas características: en Estados Unidos hay la TELENET, y en Francia la TRANSPAC.

Un aspecto importante de las redes públicas es la estandarización de los métodos de acceso. Las normas se refieren tanto al interfaz entre la red y el ordenador (o el terminal) que desea conectarse (señales, niveles eléctricos, etc.) como a los protocolos de comunicación.

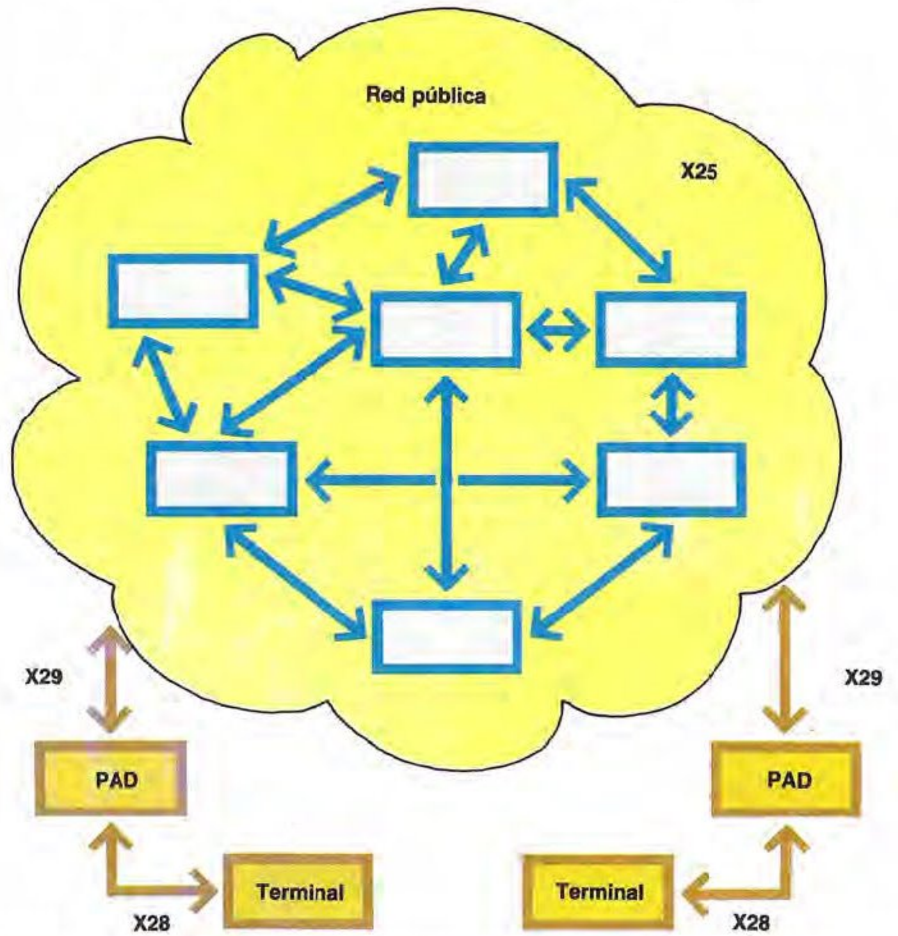
El CCITT (Comité Consultivo Internacional de Telegrafía y Telefonía), con sede en Ginebra, ha establecido la recomendación hoy más seguida en el mundo por las redes públicas, que se ha convertido en un estándar; se denomina X25 y establece las normas a las que deben atenerse los nodos de una red pública.

Los terminales (en su sentido más general) pueden conectarse a la red mediante dispositivos (compuestos de hardware y software denominados PAD [Packet Assembler-Disassembler] y reglamentados por la recomendación X3 del mismo CCITT. La conexión de un terminal al PAD está gestionada por la recomendación X28, mientras que la conexión del PAD a la red está regulada por la X29. Todos estos conceptos se resumen en la figura de la página siguiente. Las recomendaciones del CCITT han establecido un paso fundamental hacia adelante en la teletransmisión de los datos, permitiendo que los ordenadores de fabricantes diferentes se hablen entre sí. Imaginemos una sociedad que dispone en Barcelona de un ordenador de marca A, en Valencia de un ordenador de marca B y en Madrid de un ordenador de marca C. La existencia de una red pública que trabaje en X25 obliga a las empresas constructoras A, B y C a dotar a sus ordenadores de instrumentos capaces de trabajar según las modalidades de acceso a la red, y la sociedad antes citada podrá hacer comunicar los tres ordenadores sin verse obligada a formar una red privada.

Redes locales

Las redes en sentido general y, por tanto, también las redes públicas, pueden proyectarse para cubrir áreas geográficas bastante amplias. Sin embargo, también existen requisitos de distribución de las informaciones en áreas geográficas que se extienden pocos kilómetros, o a veces pocos centenares de metros. Por ejemplo, piénsese en una compañía que tiene necesidad de instalar varios ordenadores en el interior de un mismo edificio, o bien en el interior de edificios colindantes. En este caso conviene formar una red de manera que las comunicaciones entre los diversos nodos no estén basadas en el uso de las líneas telefónicas. La ventaja mayor ofrecida por las redes locales respecto a las que se extienden sobre amplias áreas es la velocidad de comunicación entre los nodos. Efectivamente, en las redes locales pueden alcanzarse velocidades de 100 millones de bits por segundo, mientras que en las otras redes no puede llegarse más allá de 50 a 60 millares de bits por segundo.

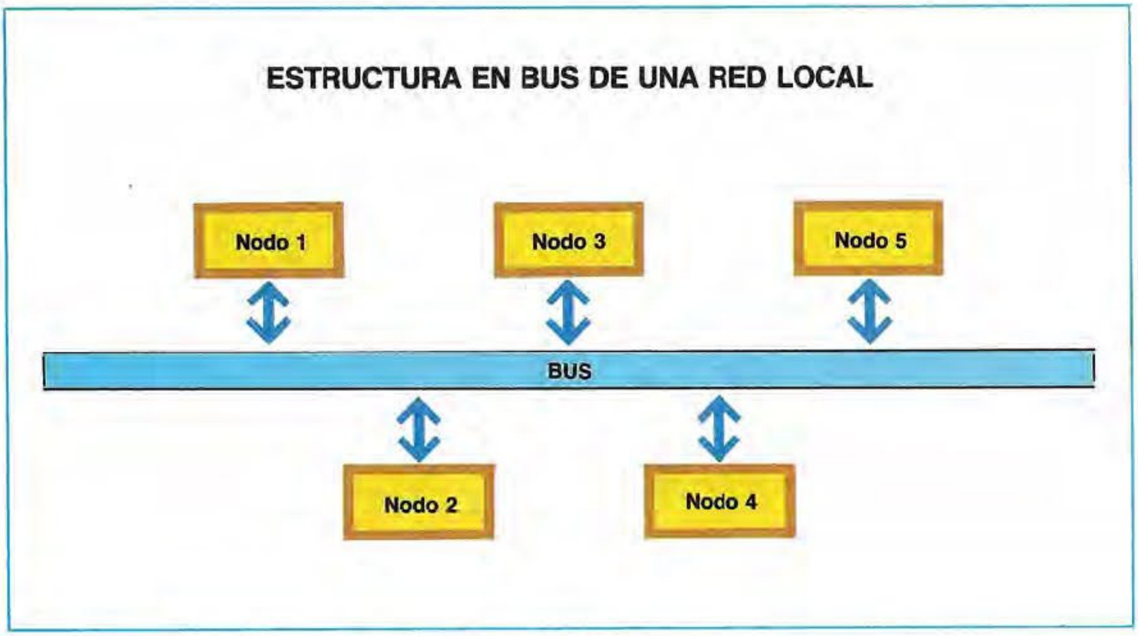
MODALIDAD DE ACCESO A UNA RED PUBLICA (RECOMENDACIONES DEL CCITT)



Las líneas de comunicación están constituidas por cables (trenzados y apantallados o coaxiales) que se instalan en el área de interés y que van a parar a derivaciones a las que están conectados los nodos. En los últimos tiempos se ha empezado a difundir el uso de las fibras ópticas como soporte físico en lugar de los cables. Las topologías de conexión para una red local (definidas precisamente con las siglas LAN, Local Area Network) son las mismas ya descritas para las redes en general (en estrella, en anillo), además con una nueva configuración, definida **de bus** y esquematizada en la página siguiente. En la estructura en bus hay la completa independencia entre nodos, en el sentido de que si uno o más nodos se averían, los otros continúan transmitiendo. Cada nodo envía un mensaje a lo largo del bus en ambas direcciones, y el nodo o los nodos destinatarios lo aceptan si van dirigidos a ellos.

El protocolo de salida para las comunicaciones en las redes locales ha sido desarrollado por el Institute of Electrical and Electronics Engineers y lleva el nombre de IEEE-802; esta recomendación establece las distancias máximas, los nodos, las velocidades, así como las modalidades de transmisión de datos para las redes locales.

Una interesante solución adoptada para las redes locales utiliza como medio de comunicación las líneas telefónicas internas de la empresa cuya sede está situada en uno o más edificios vecinos; estas líneas son gestionadas por un sistema llamado PBX (Private Branch X-



change) que permite poner en comunicación dos oficinas internas o bien derivar hacia las diversas oficinas las llamadas procedentes del exterior. Potenciando como hardware los PBX, éstos permiten la unión de las conexiones entre ordenadores por la red telefónica interna; es decir, en la misma red es posible hablar y hacer comunicar los ordenadores. Los dispositivos que pueden conectarse en una red local son los más diversos: miniordenadores y ordenadores personales, unidades de cinta o de disco, terminales de vídeo, impresoras, etc. La tendencia actual es conectar a la red muchos dispositivos de proceso y pocos periféricos, puesto que el coste de las unidades centrales va disminuyendo paulatinamente, mientras que el costo de los periféricos no baja. Por tanto, el objetivo es compartir con más unidades centrales las unidades de disco o de cinta y las impresoras. El siguiente paso de crecimiento para una red local es su posibilidad de conectarla a las redes exteriores, o bien la posibilidad de que todos los dispositivos de una LAN puedan estar conectados a una red pública, por ejemplo para tener el acceso a grandes bancos de datos y a grandes potencias de proceso geográficamente lejanas.

Arquitecturas que surgen

Todos los instrumentos adoptados en la comunicación de los datos, como líneas, protocolos, redes, etc., constituyen los ladrillos de un gran edificio cuya construcción todavía no ha terminado. Este edificio es la arquitectura general de las redes de comunicación. Actualmente existen dos grandes arquitecturas, proyectadas para unificar la comunicación de datos, que conducen a dos conceptos diferentes:

1 / System Network Arquitecture (SNA)
2 / Open System Interconnection model (OSI)

Las dos arquitecturas están construidas en capas. La primera capa, por ejemplo, dice cómo efectuar las conexiones eléctricas, cuáles son los valores de las señales, etc.; la segunda establece las modalidades y los protocolos de conexión, y así sucesivamente, hasta fijar modalidades de proyecto al nivel más elevado. Las dos arquitecturas (SNA y OSI) están constituidas por 7 capas (no siempre iguales entre las dos soluciones), algunas de las cuales todavía deben definirse en detalle.

La estructura SNA ha sido desarrollada por IBM, y constituye el modelo en que se inspirarán en el futuro las redes construidas por esta firma.

La estructura OSI ha sido desarrollada por la International Standard Organization (ISO), o sea por un instituto de estandarización. El objetivo del modelo es el de definir una arquitectura de red flexible en la que puedan inspirarse todos los fabricantes de ordenadores, muchos de los cuales ya hacen referencia al modelo OSI para el hardware y el software suministrados a sus clientes.

TEST 23

1 / La función de handshake sirve

a) para conectar dos ordenadores;
b) para sincronizar el que transmite con el que recibe;
c) para construir un protocolo;
d) para hacer comunicar a distancia dos dispositivos.

2 / Los caracteres SYN se usan

a) en el protocolo HDLC;
b) en el protocolo BSC;
c) en los protocolos asíncronos;
d) para el control de los errores.

3 / En el protocolo HDLC, el texto transmitido puede ser

a) sólo de caracteres;
b) sólo de números;
c) cualquiera;
d) sólo half-duplex.

4 / El protocolo IEEE-488 es

a) serie;
b) paralelo;
c) half-duplex;
d) para largas distancias.

5 / La línea ATN (attention) se emplea

a) en el protocolo BSC;
b) en el protocolo HDLC;
c) en el protocolo IEEE-488;
d) en ninguno de estos.

6 / El packet switching sirve

a) para conmutar dos señales sobre una red;
b) para enviar a su destino un mensaje siguiendo vías diferentes;
c) para intercambiar un mensaje;
d) para excluir un mensaje.

7 / Una red pública permite

a) sólo el uso del protocolo IEEE-488;
b) sólo la transmisión de datos en áreas limitadas;
c) la conexión de dispositivos en áreas bastante amplias;
d) el empleo de cables coaxiales.

8 / La estructura en bus se usa

a) en las redes públicas;
b) en las redes locales;
c) en el protocolo X3;
d) en las conexiones serie.

Las soluciones en la pág. 1367.

Instrucciones del Basic para la comunicación de datos

También en el lenguaje Basic, que es el más difundido en los microordenadores y ordenadores personales, permite la gestión de la comunicación de datos.

Básicamente no existen «instrucciones estándar» para este propósito, y a continuación ilustraremos principalmente a título de ejemplo, una metodología de empleo de las instrucciones para resolver problemas de comunicación asíncrona en RS232.

La activación de una comunicación pasa por las siguientes fases principales:

— apertura de comunicación del canal con una instrucción del tipo OPEN
— lectura o escritura de los caracteres en el canal previamente abierto

La lógica que se sigue es la de asimilar el canal de comunicación a un file y, por tanto, realizar las operaciones de I/O como si se tratase de un acceso al file.

La instrucción OPEN reserva un buffer de I/O del mismo modo que si se tratase de un file. Generalmente, los parámetros a proporcionar son los siguientes:

N Número del canal. A veces coincide con el número de una puerta I/O o de un adaptador

V Velocidad de transmisión (bits por segundo). Acostumbran a estar previstas las velocidades 75, 110, 150, 300, 600, 1200, 1800, 2400, 4800, 9600 bps; la más empleada es la de 1200 bps

P Paridad. Indica el método utilizado para el control de paridad en el carácter. Se han previsto los siguiente métodos:
— ausencias de control; el bit de paridad no se transmite
— ausencia de control con bit de paridad transmitido siempre con el valor 0
— ausencia de control con bit de paridad transmitido siempre con el valor 1
— control de paridad par
— control de paridad impar
Esta elección de posibilidades podría parecer muy amplia, ya que incluye casos particulares que difícilmente se encuentran en la práctica. En realidad debe considerarse que muy a menudo deben acoplarse dos o más máquinas de constructores diferentes, controladas por sistemas operativos diferentes y programadas con lenguajes diferentes. En estos casos, la configurabilidad completa del interfaz es una condición esencial para la buena consecución de la conexión.

D Número de bits que constituyen un carácter (excluido el bit de paridad). Los valores admitidos son 4, 5, 6, 7, 8; el más empleado es el 7

S Número de bits de paro; los valores más usados son 1 y 2

F Número del file lógico asociado al canal; deberá utilizarse en las siguientes operaciones de I/O

C Número máximo de bytes previsto en el buffer de comunicación

Por ejemplo, la instrucción

OPEN 'COM 3: 1200, S,7,1' AS 1,128

abre un file de comunicación con los siguientes parámetros:

- número del canal: 3
- velocidad de transmisión: 1200 bps
- bit de paridad siempre en 1 (se ha supuesto que se puede indicar esta opción con la letra S; en la realidad debe consultarse el manual del ordenador utilizado)
- número de bits que forman un carácter: 7
- número de bits de paro: 1

La palabra reservada AS seguida de los parámetros 1,128 asigna este canal al file 1 con un buffer de 128 caracteres.

Además de estos parámetros principales, en algunos casos debe gestionarse también el estado de las señales. En particular, si en la línea se ha previsto un modem, puede ser necesario implantar algunos indicadores que especifiquen el estado de algunas líneas (activo/no activo). Las principales líneas utilizadas en este tipo de comunicación son:

- Request to Send
- Clear to Send
- Data Set Ready
- Carrier Detector

Su uso depende del tipo de hardware de que se dispone y de la estructura software. En algunas aplicaciones pueden utilizarse, por ejemplo, para generar interrupciones que activan una subrutina de adquisición de un carácter. Si está sometido a este tipo de software, después de haber implantado los parámetros, el ordenador se pone en un bucle de espera, durante el cual examina continuamente el estado de una deter-

Conducción de luz láser en un haz de fibras ópticas.



Italtel

minada señal que indica conexión activa (por ejemplo el Carrier Detector). Este bucle puede abandonarse únicamente de dos maneras:

— a continuación de una interrupción
— por caída de la señal

La interrupción, activada por la tarjeta del interfaz, indica la llegada de un carácter. El bucle se abandona provisionalmente y se activa la rutina de adquisición. Al final se recupera el flujo del programa tal como estaba antes de la interrupción (es decir, se reactiva el bucle de espera) hasta la llegada del siguiente carácter o hasta la caída de la señal bajo control. En este segundo caso (fin de la conexión), el programa termina. Este tipo de gestión no es tan sencillo como puede parecer; ante todo puede estar adoptado sólo en algunos tipos de máquinas y en determinadas configuraciones; además, a nivel software se complica por la necesidad de accionar toda una serie de controles y verificaciones. Los principales controles a efectuar pueden relacionarse así:

— timeout
— llenado del buffer
— paridad
— prioridad de las interrupciones
— caída de la línea u otras interrupciones accidentales

Se habla de timeout cuando entre un carácter y otro transcurre un intervalo de tiempo superior al máximo previsto: el software debe poder detectar esta situación anómala y decidir en consecuencia cuál es la decisión que debe tomar. Normalmente suele emitirse un mensaje de diagnóstico y se interrumpe la conexión. El llenado del buffer, en particular en el Basic interpretado, puede producirse cuando el programa debe realizar demasiados procesos en la subrutina de adquisición. En esta subrutina, la interrupción debe estar necesariamente desactivada; de otro modo, a la llegada de un carácter, la rutina se llamaría a sí misma, a veces muchas veces consecutivamente. En esas condiciones, la rutina no puede ser interrumpida y, si su desarrollo necesita tiempos muy largos, los caracteres que van llegando, al no ser aceptados, llenarían el buffer de comunicación hasta generar un error, o bien causando un funcionamiento defectuoso del programa.

La manera más sencilla de obviar este inconveniente consiste en reducir al mínimo los procesos de la rutina de adquisición, pero también esta solución presenta incógnitas; por ejemplo, eliminando los controles se corre el riesgo de aceptar como dato válido un valor erróneo.
También existen interfaces que pueden realizar el control sobre los caracteres recibidos. En caso de error, estos interfaces envían al ordenador un código particular que debe interpretarse en la rutina de adquisición, en la que siempre deben insertarse por lo menos los controles principales. El control de paridad es otro punto crítico del software de gestión de las comunicaciones. Normalmente, este control lo realiza el hardware de manera completamente transparente. El programa queda informado de la existencia de un error con la llegada de un código particular o con la activación de una señal adecuada. En este segundo caso, además de la utilizada para la llegada de los datos, es necesario activar otra línea de interrupción, con las siguientes complicaciones gestión de las prioridades.
En general, en el momento de la generación, las interrupciones producen una reserva de servicio. Por tanto, se crea una cola de peticiones que se agota gradualmente, a medida que el sistema cumple con las principales prioridades. El desarrollo del fenómeno, del todo asíncrono, no está ligado de ninguna manera a las temporizaciones del ordenador; por tanto, la puesta en cola de las diferentes interrupciones no influye en el flujo de los datos de llegada, que se acumulan en el buffer. Si el sistema está dedicado a resolver tareas de prioridad principal, no puede tomar los datos del buffer y, en consecuencia, se produce un fenómeno análogo al llenado del buffer debido a la excesiva longitud del proceso. Esta causa todavía es más difícil de eliminar, puesto que se trata de una causa exterior al flujo normal del programa.
Normalmente, el problema se resuelve asignando al interfaz de comunicación la máxima prioridad, estableciendo así que éste debe ser atendido en primer lugar.
Por último, debe considerarse la eventualidad de una avería que bloquee la señal al nivel activo incluso cuando la transmisión se ha terminado. Si esto sucediese, el bucle de adquisición, que utiliza esta señal como indicador de estado, no quedaría nunca interrumpido y el ordenador permanecería en espera sin proporcionar ningún mensaje de diagnóstico.

SOLUCIONES AL TEST 23

1 / b

2 / b

3 / c

4 / b

5 / c

6 / b

7 / c

8 / b

Esta eventualidad puede incluirse en la gestión del timeout, previendo un oportuno lapso de tiempo, muy superior al necesario para recibir un carácter, transcurrido el cual el programa deberá terminar (por timeout).

Como aparece evidente de la problemática expuesta, para gestionar correctamente un interfaz de comunicación es necesario disponer de un temporizador, o sea de un reloj que pueda señalar el paso del tiempo.

Las instrucciones relacionadas con la gestión del reloj tienen una sintaxis ligada a la presencia o no de un hardware determinado; por tanto, las formas posibles dependen mucho del tipo de máquina utilizada. Una forma muy usada es:

TIME$ = A$ para la implantación del tiempo
B$ = TIME$ para la lectura del tiempo

Generalmente,el valor del tiempo se expresa en forma de una cadena de 6 caracteres, que contiene horas, minutos y segundos en el formato HH MM SS.

La primera instrucción implanta el tiempo al valor indicado en A$; la segunda lee el valor en curso como diferencia entre el valor implantado y el valor leído.

La gestión del timeout puede obtenerse controlando el tiempo transcurrido en la rutina de adquisición; si éste resulta superior a un valor máximo prefijado, se activa un flag de error y se termina el programa; de otra manera se pone a 0 el temporizado disponiéndolo para el siguiente control. De este modo se mide el intervalo de tiempo que transcurre entre la llegada de un carácter y otro, pero este control no puede ser suficiente para verificar todas las posibilidades de causa de timeout. Deberá ser el programador el que establezca qué otros puntos del programa son críticos y, en consecuencia, prever para ellos un control análogo.

A pesar de las limitaciones de que se ha hablado, el lenguaje Basic es todavía el único (aparte del Assembler) que permite este tipo particular de aplicación dentro de la categoría de los microordenadores. Los otros lenguajes no prevén en general ninguna instrucción para la comunicación de datos.

Sistemas operativos

Las actividades que se realizan en un ordenador personal están regidas por el intérprete Basic y deben predisponerse en el programa respetando las reglas que impone el lenguaje. Por tanto, el Basic puede imaginarse como un gran programa que da la posibilidad a quien lo usa de aprovechar de manera eficiente el ordenador. Generalizando este concepto, se llega a la definición de **sistema operativo**: un conjunto de programas y de procedimientos que permite la utilización de los recursos del ordenador.

También si el ordenador es de tipo personal, no siempre el Basic basta por sí solo para coordinar las actividades y, por tanto, es necesario utilizar sistemas operativos como el CP/M, el MPM y otros. En general, un mismo ordenador puede albergar sistemas operativos diferentes, cuya intervención está ligada a las necesidades de la aplicación particular. El que deba desarrollar programas que hagan un uso intensivo de files elegirá, por ejemplo, un sistema operativo que permite la multiprogramación. Si bien en general es verdad que un ordenador puede albergar sistemas operativos diferentes, en la práctica, el ordenador nace con el sistema operativo puesto a punto por la firma constructora.

En la definición de sistema operativo se ha hablado de **recursos**. Con este término se hace referencia a las características y propiedades de un sistema de proceso: la memoria central, las memorias masivas, el tiempo de CPU, la capacidad de reacción a estímulos externos, etc. Por tanto, decir que el sistema operativo es el controlador de estos recursos también significa evaluar cómo los coordina, o bien si consigue gestionar un determinado recurso interesado en una particular aplicación. Para esto analizaremos las principales características de los sistemas operativos, puntualizando el significado de la terminología que acompaña a las mismas. A continuación se hará una referencia continua al sistema operativo UNIX, que actualmente tiende a ser el más difundido y empieza también a ser implantado en los ordenadores personales. El UNIX ha sido desarrollado por Bell Laboratories y se está afirmando en las universidades norteamericanas por su simplicidad de uso y por su carácter «amistoso». Actualmente, todas las firmas constructoras de ordenadores tienden a instalarlo en sus minis y, por tanto, este sistema operativo se está convirtiendo de hecho en el vehículo de transporte del software sobre máquinas de firmas diferentes.

Monousuario y multiusuario

Con el término usuario suele entenderse el número de los puntos de acceso al ordenador, y esta característica suele indicar cuántas personas o usuarios pueden utilizar simultáneamente un ordenador. A los puntos de acceso corresponden terminales vídeo y teclados. En el **monousuario** se tiene un solo terminal vídeo, que en los personales coincide con el propio ordenador, mientras que el **multiusuario** es una característica típica del microordenador y, evidentemente, de los mainframes. Los sistemas operativos deben coordinar el acceso al ordenador de los diversos usuarios, repartiendo el tiempo de CPU, la memoria, los files y también permitiendo el uso del sistema solamente a los usuarios autorizados. El número de personas cualificadas como usuarios está prefijado, y el acceso de cada usuario está vinculado a la introducción de contraseñas conocidas solamente por las personas autorizadas. El acceso (**login**) a un ordenador controlado por el UNIX se ha esquematizado en la figura de la página siguiente. El usuario pulsa una tecla cualquiera del terminal y el sistema responde con la cadena «;login:». El usuario pone su firma introduciendo su apellido y el sistema pide la «password». El ordenador ya ha definido previamente la contraseña para el usuario «perez» que corresponde a la palabra «jose»; si como contraseña se introduce «juan», el UNIX niega el acceso presentando el mensaje «Login incorrect». Volviendo a pulsar una tecla cualquiera reaparecerá la leyenda «;login:» como invitación a volver a firmar. Introduciendo la contraseña correcta se tendrá el acceso al sistema; el UNIX presentará la fecha en la que «perez» ha efectuado la última conexión y el carácter $, que significa «estoy preparado para aceptar instrucciones de perez». Las contraseñas introducidas no se visualizan, para evitar que eventuales usuarios no autorizados puedan conocerla.

ACCESO AL UNIX

Presentado por el ordenador | Introducido por el usuario | Introducido por el usuario y no presentado

```
; login:  perez
password:  juan
Login incorrect
; login:  perez
password:  jose
Last login: Mon Jun 4-10:32
```

Batch e interactividad

Los sistemas operativos **batch**, que por largo tiempo han constituido la única modalidad de trabajo en ordenadores y preveían trabajar con el esquema de «una actividad cada vez», ahora están en discusión. Todas las actividades que debía realizar la máquina (tareas) se ponían en cola secuencialmente, y el sistema operativo procedía a atenderlas por turno. La modalidad de gestión era del tipo First In/First Out (FIFO), en el sentido que tenía la preferencia la tarea cargada en primer lugar. Los sistemas operativos interactivos se basan en la no predeterminación de lo que el usuario va a pedir al ordenador y controlan terminales de vídeo y teclados, que constituyen el medio interactivo más usado. En el vídeo aparece siempre un carácter (o una cadena) de espera con el que el sistema operativo informa al usuario de la apertura de la línea. El UNIX es un sistema interactivo y su señal de espera es el carácter $. También los ordenadores personales están guiados por procesos interactivos; el propio Basic o el CP/M tienen un carácter interactivo.

La presencia en un ordenador de un sistema operativo (batch o interactivo) implica también el concepto de que este último tiene el control de las operaciones que realiza la máquina. Imagínese por ejemplo que un programa en ejecución efectúe una operación ilícita: el sistema operativo debe verla, señalarla y reasumir el control. En un sistema batch, reasumir el control significa abandonar la tarea en ejecución y pasar a la siguiente; en un sistema interactivo significa recuperar la señal de espera en el vídeo después de la oportuna señalización.

Multiprogramación

La multiprogramación es una característica que puede poseer una determinada máquina o no, pero que suele estar incluida en todos los sistemas operativos que gestionan la modalidad de multiusuario, porque los usuarios que acceden al ordenador desde puntos diferentes, por norma, desean ejecutar programas diferentes.

En principio, la multiprogramación se hace necesaria para aprovechar los tiempos muertos de la máquina. Supongamos que un operador esté introduciendo datos para un programa a través del teclado de un terminal de vídeo; el programa no podrá proseguir la ejecución hasta que todos los datos que se espera recibir se hayan introducido. El tiempo necesario para introducir estos datos es extremadamente largo si se compara con los tiempos típicos de un procesador. Por tanto, si el sistema operativo lo permite, la máquina puede realizar la ejecución de otro programa mientras el operador introduce los datos. Se entiende por **multiprogramación** la capacidad que tiene un sistema operativo para gestionar simultáneamente varios programas presentes en el ordenador. Gestionar no significa realizar simultáneamente; el significado de gestión se entiende como capacidad de aprovechar los tiempos muertos suspendiendo temporalmente la ejecución de un programa para dar la posibilidad de que se ejecute otro. Los sistemas operativos que permiten la multiprogramación deben gestionar un orden de prioridad para asignar cada programa. Supongamos que la prioridad sea un número entero comprendido entre 1 y 99, y que el nivel de prio-

ridad sea decreciente: los programas de prioridad 20 son más importantes que los programas de prioridad 40, que a su vez son más importantes que los de prioridad 60 y así sucesivamente. Por norma, en la máquina existen varios programas que piden su realización. Estos programas forman una cola que acostumbra a definirse como **lista de agenda**. El sistema operativo debe decidir qué programa debe realizar en primer lugar. Un ejemplo significativo se esquematiza en la figura de abajo. Hay tres programas que piden que se ejecuten: A con prioridad 30, B con prioridad 20 y C con prioridad 10. La descripción de lo que sucede en el período entre T_1 y T_{10} es la siguiente:

T_1 La máquina empieza a realizar A, porque en este momento es el único que hay en el ordenador

T_2 El programa B (más importante que A) pide que se realice; el sistema operativo suspende A y empieza B

T_3 El programa C (más importante que B) pide ser atendido; el sistema operativo suspende B y empieza C

T_4 C realiza una operación lenta (por ejemplo entrada de datos por terminal); el sistema reemprende la ejecución de B

T_5 Termina la fase lenta de la ejecución de C; B se suspende y se reemprende C

T_6 C ha terminado su ejecución; el sistema reemprende B

T_7 B realiza una operación de entrada/salida de datos; el sistema reemprende A

T_8 Termina la fase de entrada/salida de B; A se suspende y se reemprende B

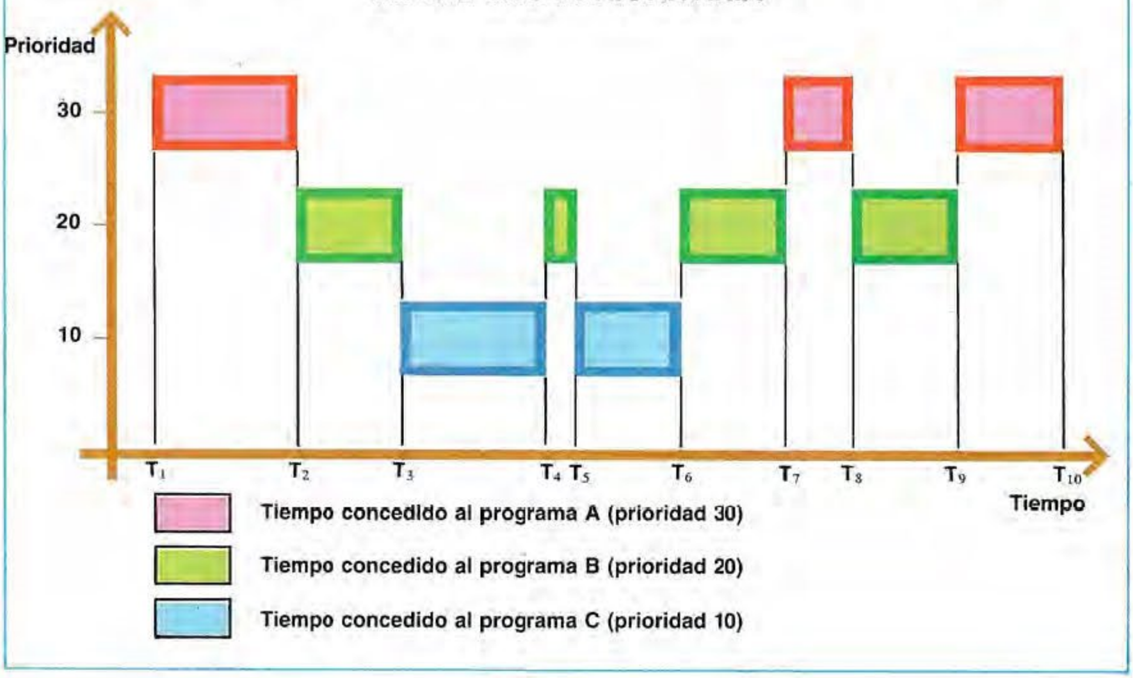
T_9 Termina la ejecución de B, y el sistema reemprende la de A

T_{10} Termina también la ejecución de A

Como puede verse, la multiprogramación en base prioritaria permite también establecer una jerarquía sobre la importancia de los programas. El ejemplo descrito puede asociarse a un ordenador con tres terminales, en cada uno de los cuales hay un operador que envía a ejecución uno de los programas A, B o C.
También puede suceder que la máquina esté realizando un programa X de prioridad 50 y otro programa Y, también de prioridad 50, pide ser atendido. Para gestionar esta situación existen dos modalidades:

1 / el programa X debe terminar antes de que se inicie Y, a menos que X no entre en una operación lenta, en cuyo caso se atiende Y

MULTIPROGRAMACION. EJECUCION DE TRES PROGRAMAS SOBRE BASE PRIORITARIA



2 / los programas X e Y se envían a ejecución alternativamente en intervalos de tiempo de duración prefijada

Esta segunda eventualidad se describe a continuación.

Tiempo compartido

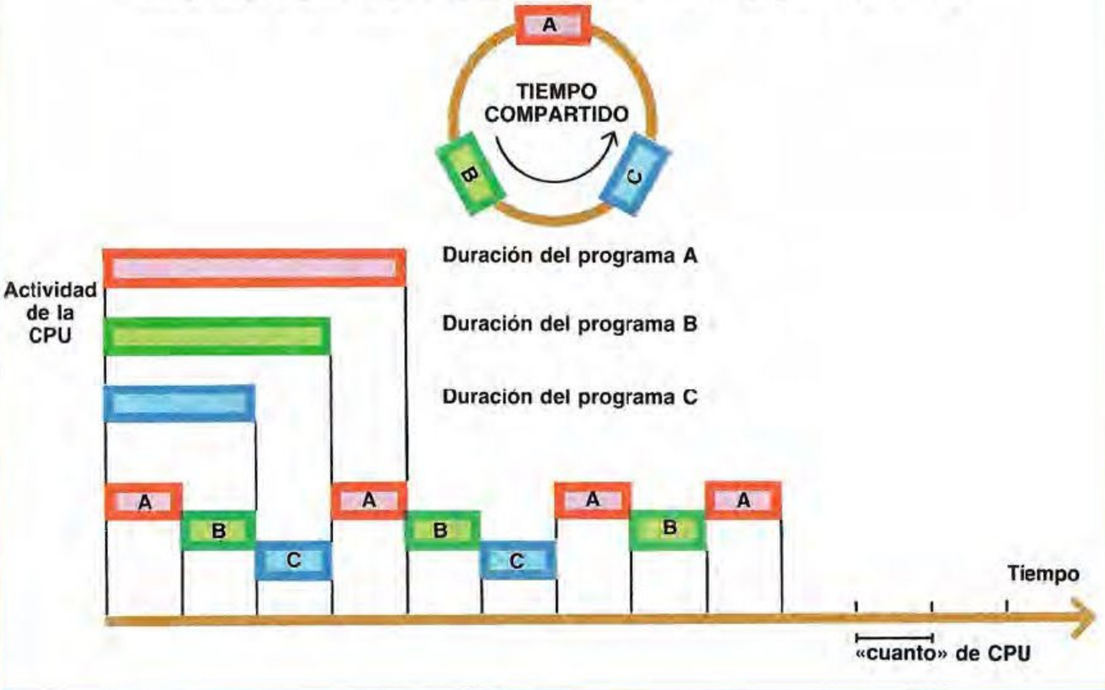
Realizar programas en **tiempo compartido** significa asignar a cada uno de ellos un intervalo de tiempo muy bien determinado. Esto significa que la CPU del procesador dedica su atención a un determinado programa sólo durante el tiempo preestablecido, después pasa a otro programa, después a otro y así sucesivamente hasta que reemprende el programa inicial y recomienza el ciclo. El intervalo temporal se acostumbra a llamar **cuanto de CPU,** y sobre esta base la gestión de los programas se realiza de manera circular.

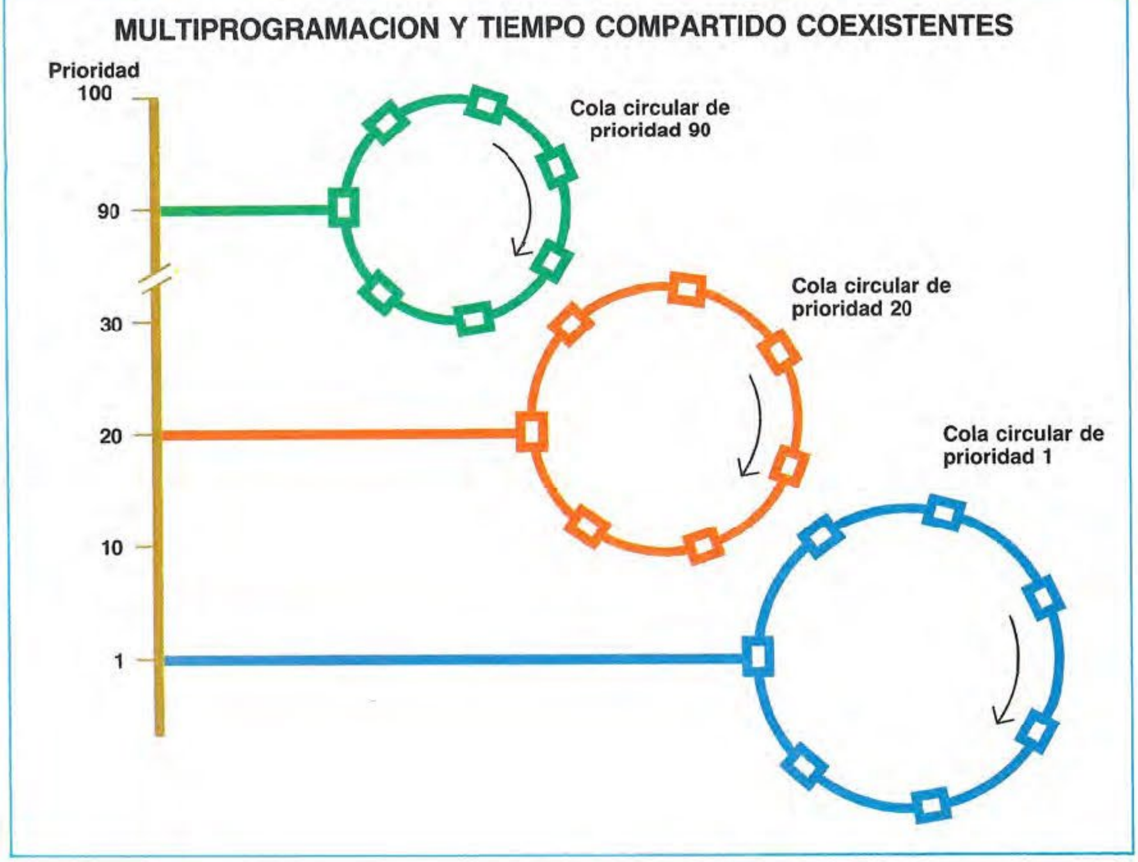
En la figura de abajo se ha esquematizado la gestión de tres programas, A, B y C, en tiempo compartido. En este caso no interviene ningún concepto de prioridad asociada a los programas; A, B y C tienen el mismo derecho de ser realizados, y la CPU del procesador contenta a todos concediéndole a cada uno por turno un cuarto de tiempo.

En algunos sistemas operativos, la multiprogramación y el tiempo compartido coexisten formando una estructura muy flexible para la coordinación de programas en competencia. La situación se ha esquematizado en la figura de la página siguiente. En teoría hay tantas colas circulares como cuantos son los valores posibles de la prioridad 1, después pasará a la cola de prioridad 20 y, finalmente, a la cola de prioridad 90. En esta situación, el usuario puede diversificar al máximo la importancia de los programas que puede escribir y, viceversa, en el sistema operativo es posible establecer clases diversificadas de usuario. Es decir, es posible decidir que un grupo de usuarios divida el tiempo de CPU en la prioridad 50, otro grupo en la prioridad 20 y así sucesivamente.

El sistema operativo UNIX trabaja en multiprogramación y tiempo compartido, pero el usuario no puede establecer la prioridad de un programa; todos los programas, cuando empiezan la ejecución, parten de la misma prioridad, por ejemplo 10, y el UNIX eleva el valor de la prioridad (reduciendo su preferencia) si el programa tiende a durar. Este mecanismo penaliza los programas que duran demasiado y agiliza los que duran poco; su objeto es optimizar al máximo la interactividad del sistema operativo.

GESTION DE TRES PROGRAMAS EN TIEMPO COMPARTIDO





Tiempo real

El funcionamiento de la multiprogramación y del tiempo compartido se basa en el concepto de interrupción de la actividad que realiza normalmente un procesador. Si hay en ejecución un programa de prioridad 50 y pide el uso de la CPU un programa de prioridad 40, el sistema operativo debe interrumpir la actividad en curso (ejecución del programa de prioridad 50) y activar la nueva actividad. La función se realiza asociando al programa que pide ser atendido (o en general a cualquier programa que pide el uso de la CPU) una interrupción al sistema operativo. Por tanto, este último deberá tener una arquitectura interna basada en la posibilidad de aceptar interrupciones desde el exterior. La interrupción puede proceder de cualquier punto de acceso al procesador y, por tanto, a menudo de cualquier periférico conectado, sea éste tradicional (terminal, impresora, disco, cinta, etc.) o no. Se entiende por periférico no tradicional por ejemplo, un instrumento de medida o un sensor (detector). Imaginemos que se quiere controlar con el ordenador un proceso industrial (una refinería, una fábrica de automóviles, etc.). Entonces, al ordenador estarán conectados directamente dos conductores por los que viajan las señales eléctricas procedentes de los sensores del proceso; en base a los valores de estas señales, el ordenador debe decidir si cerrar una válvula, hacer girar un motor, excitar un relé, etc. En este caso, la acción que el ordenador realiza hacia el exterior debe ser lo más rápida posible, y por tanto debe intervenir en un tiempo pequeño con respecto a la duración del proceso. El sistema operativo que controla el ordenador para estas aplicaciones se define como **tiempo real**. La característica de base de un sistema operativo de tiempo real es una arquitectura profundamente basada en la gestión de las interrupciones con una elevada velocidad de respuesta. La velocidad con que el sistema operativo realiza la interrupción permite al ordenador controlar procesos igualmente rápidos; cuanto más rápida es la ejecución, tanto más rápido puede ser el proceso controlado. En los sistemas operativos de tiempo real, la instrucción ex-

terior puede asociarse también a programas escritos por el usuario; es muy típica la situación en la que el cierre de un contacto eléctrico corresponde a la ejcución de un programa que el usuario ha previsto para gestionar este acontecimiento. El sistema operativo UNIX no es de tiempo real y no permite la conexión de periféricos no tradicionales, ni proporciona la posibilidad de gestionar interrupciones.

Sistema file

El sistema file identifica la modalidad con que un sistema operativo gestiona las memorias de masa, y entre éstas, particularmente los discos. Un ordenador no puede utilizar las memorias no volátiles, porque en éstas se conservan todas las informaciones de carácter permanente de las que tiene necesidad. Las memorias de masa también contienen el sistema operativo, que se carga automáticamente en la memoria central al poner en marcha el procesador. Por ejemplo, en algunos ordenadores personales, el sistema operativo reside en discos rápidos (discos rígidos) y desde ellos se carga en memoria siguiendo procedimientos especiales (definidos como start-up) que varían en cada caso. Los mismos discos contienen también informaciones depositadas por los usuarios y el sistema file cuida de gestionarlos con las modalidades más eficientes.
Los files y los records en los discos pueden ser de naturaleza diferente, y pueden clasificarse de diversas maneras.

- **Clasificación en base al contenido**. Los records pueden contener datos ASCII (o EBCDIC) o binarios; los primeros se refieren a programas fuente o textos, los segundos a los resultados de compilaciones (files reubicables), a los programas realizables o a los archivos de datos calculados.
- **Clasificación en base a las modalidades de acceso**. Se tendrán files secuenciales y de acceso aleatorio. Los primeros se distinguen por el hecho de que para acceder a un record es necesario recorrer todos los records anteriores; en cambio, los segundos pueden direccionar un record de modo directo.
- **Clasificación en base al tipo de record**. Se

Sistema de proceso de datos que comprende un ordenador personal conectado a un ordenador huésped vía teléfono.



Marka/Photri

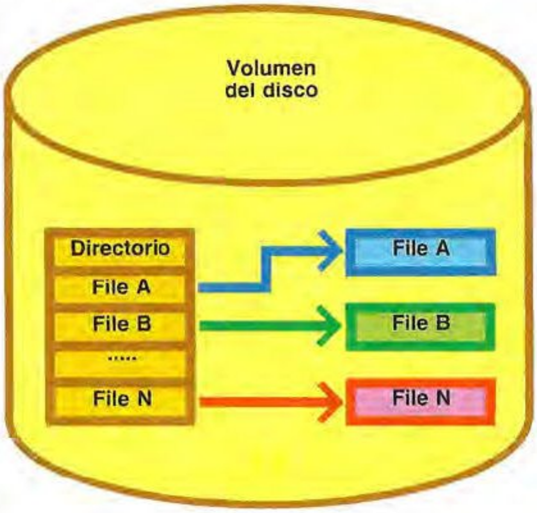
tendrán files con records de longitud fija o variable. La longitud fija distingue, por ejemplo, los programas fuente de los textos en general.

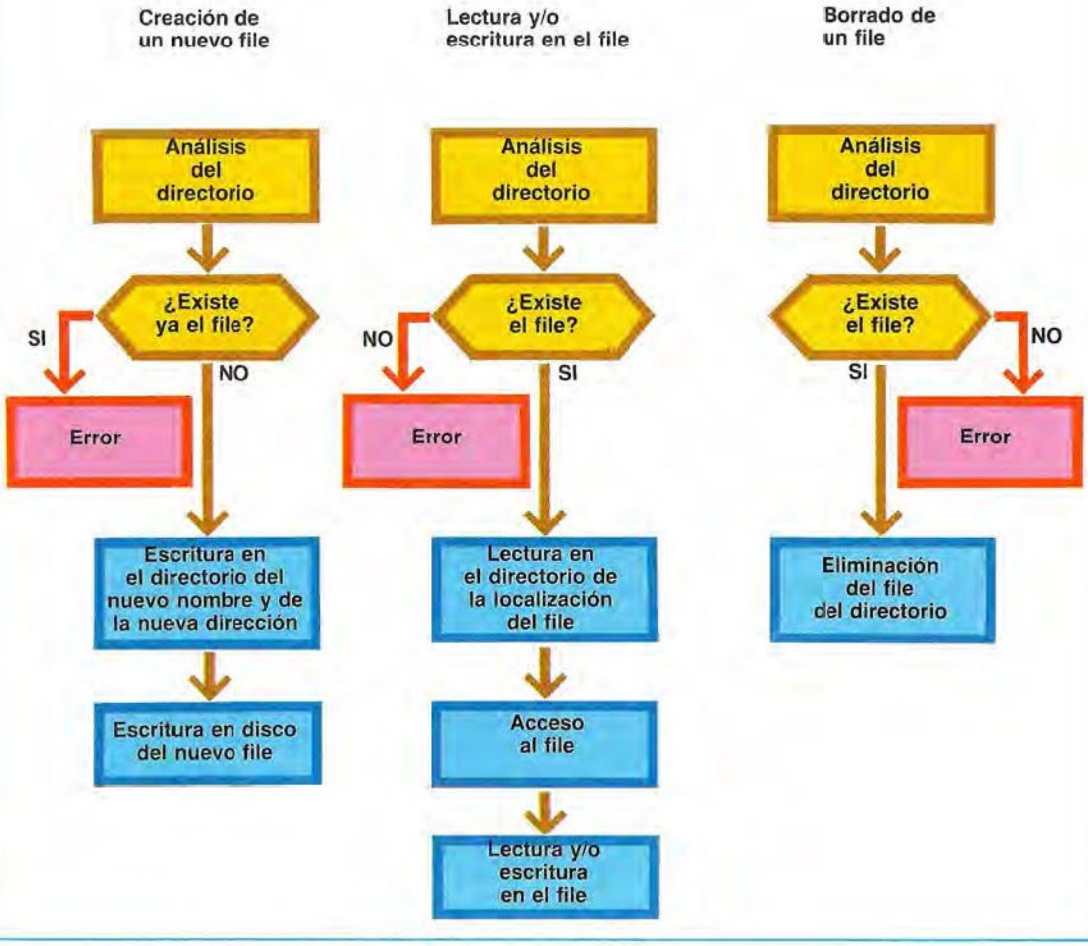
El sistema operativo tiene la misión de proporcionar al usuario la posibilidad de gestionar files de cualquier tipo, y en un ambiente de multiusuario, garantizar al usuario X que sus files no serán alterados o borrados por el usuario Y.
Un aspecto importante corresponde además a la posibilidad por parte de varios usuarios de acceder al contenido de un mismo file; en este caso, el file es el recurso que debe coordinarse garantizando la integridad de su contenido. Por ejemplo, suponiendo que el usuario X esté escribiendo un record y que este mismo record lo esté leyendo el usuario Y, las dos operaciones deben separarse, en el sentido de que mientras el usuario X esté escribiendo, el usuario Y no pueda leer. Asociando el concepto de recurso a la cantidad total de memoria de los discos, se tiene otro interesante aspecto del sistema file de un sistema operativo: la división de esta memoria entre los usuarios.
Los modernos ordenadores personales pueden tener en línea también más discos, con una capacidad total de miles de megabytes. Fijados en la configuración hardware, los megabytes totales disponibles, debe decidirse cómo repartirlos entre los usuarios. La división más espontánea tendería a repartir, por igual el espacio a disposición de los usuarios, pero este método no satisface todas las exigencias: puede suceder por ejemplo que se disponga de un gran espacio y se necesite menos.
Los sistemas operativos modernos conceden espacio en el disco a medida que se necesita, evitando por tanto que un solo usuario lo emplee todo. Por otra parte, cada usuario tiene unos límites precisos que no debe superar.
Una vez concedido el espacio en el disco, el usuario debe tener la posibilidad de acceder a sus propios files de manera rápida y transparente. Este objetivo se consigue empleando el concepto de directorio de cada volumen en que está dividido el disco. Se entiende por volumen una división lógica (y no física) del disco, correspondiente a varias aplicaciones o varios usuarios. Esta división en volúmenes también puede ser más elaborada; por ahora sólo nos referiremos al volumen como una parte de un disco que tiene por lo menos un directorio propio. El directorio es una zona del disco en la que se escriben todos los nombres de los files registrados, así como notas acerca de la estructura de los files y records, y sobre todo, la dirección del file en el disco (ver la fig. de la pág. siguiente). Cada vez que se crea un nuevo file, la presencia de éste último se registra antes que nada en el directorio y después en la zona del disco que debe contenerlo físicamente. Si se quiere escribir informaciones en el nuevo file, el sistema operativo controla el directorio y, una vez identificado el file examinado, determina su posición en el disco y después se tiene acceso de modo directo. Lo mismo sucede si se quiere leer el contenido de un file. Si finalmente se quiere conocer cuáles son todos los files depositados en el disco, basta con leer el directorio e imprimir los nombres encontrados.

Estructura sistema file del UNIX

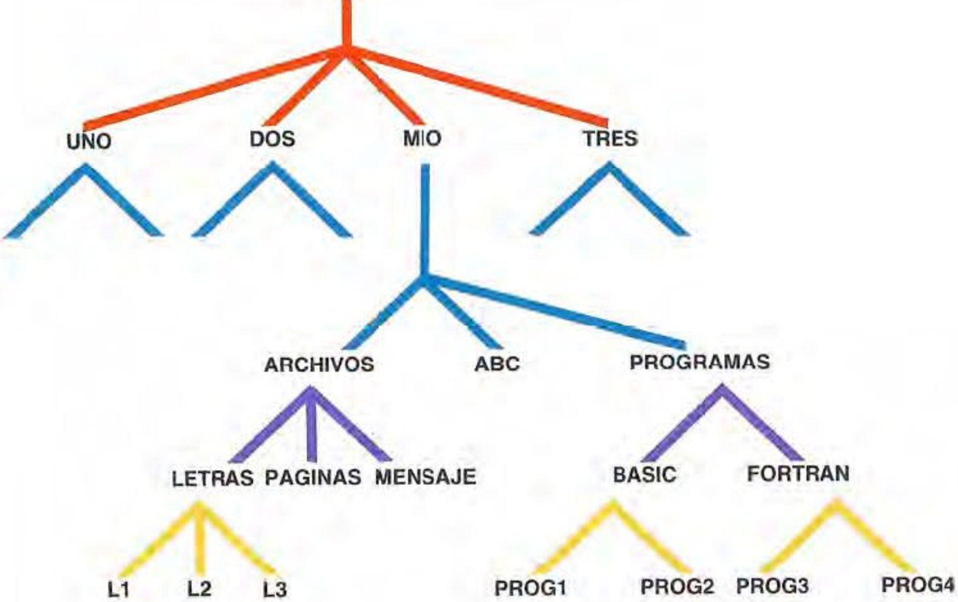
El sistema operativo UNIX amplía el concepto de directorio en el sentido de que en el disco existen files y directorios enganchados entre sí. Un directorio es visto por el UNIX como un file que puede apuntar a otros files o a otros directorios; el directorio apuntado se define como **subdirectorio** y puede apuntar a su vez a otros subdirectorios o a otros files, y así sucesivamente. Por tanto, la estructura del sistema file es del tipo de árbol, como se ha esquematizado en la figura de la página 1376. El símbolo / (barra) identifica en el UNIX el directorio raíz que da principio a toda la ramificación; de éste parten, por ejemplo, los directorios UNO, DOS, MIO, TRES (un directorio está identificado por el hecho de tener por lo menos una ramificación). El directorio MIO apunta a dos subdirectorios (ARCHIVOS y PROGRAMAS) y a un file (ABC); ARCHIVOS apunta a otro subdirectorio (LETRAS) y a dos files (PAGINAS y MENSAJE); LETRAS apunta a tres files (L1, L2, L3). Lo mismo vale para el subdirectorio PROGRAMAS.
Para el UNIX, un file es una terminación, o sea un punto del que no parte nada (una hoja del árbol), mientras que un directorio da origen a una estructura que permite agrupar desde el punto de vista lógico otros elementos (ARCHIVOS contendrá files de datos y PROGRAMAS contendrá programas). Para acceder a un file debe proporcionarse al UNIX el recorrido (path) a seguir a lo largo de las ramificaciones del árbol, que se construye a partir del directorio raíz y siguiendo el camino de acceso al file.

ACCESO AL DISCO CON USO DEL DIRECTORIO





ESTRUCTURA DEL SISTEMA FILE Y CAMINOS DE ACCESO A LOS FILE DEL UNIX



Partiendo de /

Path para MENSAJE: /MIO/ARCHIVOS/MENSAJE

Path para PROG2 : /MIO/PROGRAMAS/BASIC/PROG2

Partiendo de /MIO

Path para MENSAJE: ARCHIVOS/MENSAJE

Path para PROG2 : PROGRAMAS/BASIC/PROG2

Cada usuario habilitado para trabajar con el UNIX posee un directorio de trabajo. Por ejemplo, si el usuario Pérez está asociado previamente al directorio MIO, Pérez se posiciona autómaticamente en MIO apenas se ha obtenido el acceso al ordenador. Esto significa que para identificar los files MENSAJE y PROG2 ya no es necesario partir del directorio raíz, sino que basta con especificar por ejemplo para PROG2: PROGRAMAS/BASIC/PROG2. En este caso, el camino no empieza con la barra propia por el hecho de que no se está partiendo del directorio raíz, sino del subdirectorio MIO (directorio de trabajo para el usuario Pérez). El directorio de trabajo puede cambiarse con un comando del sistema operativo, por lo que para Pérez es posible posicionarse, por ejemplo, directamente en BASIC y acceder a los files PROG1 y PROG2, especificando solamente su nombre.

En la figura de la página siguiente se ha descrito un diálogo usuario/UNIX con algunos comandos que interaccionan con el sistema file. Para saber en qué directorio se encuentra el usuario, basta que éste introduzca el comando pwd

(print working directory): el sistema operativo responde escribiendo el camino que, a partir del directorio raíz, llega al directorio de trabajo. El comando ls presenta los nombres de los files incluidos en el directorio de trabajo (para /MIO sólo ABC, puesto que ARCHIVOS y PROGRAMAS son a su vez directorios); para conocer todos los nombres de los files directorios apuntados por /MIO basta con añadir la opción a (all) al comando ls (las opciones deben ir precedidas del carácter –). La respuesta del sistema operativo contiene todos los nombres de los files apuntados por /MIO, y para más información debe usarse el comando file que responde informando si el nombre corresponde a un directorio o a un file.
El comando cd (change directory) permite desplazar el punto de trabajo y, en el ejemplo, apuntar directamente al directorio FORTRAN. En la misma línea pueden introducirse más comandos separados por el carácter ;. En nuestro caso se ha añadido ls-a para tener en un solo comando el cambio del directorio y la lista de los files y de los directorios contenidos en el nuevo directorio.
El comando cat (catalog) permite el acceso al contenido de un file y, en el ejemplo, proporciona la lista de las instrucciones que componen el programa PROG3. El comando mkdir (make directory) permite la creación de un nuevo directorio apuntado para FORTRAN. Después de mkdir, la lista de los nombres apuntados por FORTRAN también contiene SUBR y el comando file que dice que se trata de un directorio.
Como puede observarse, cada usuario en el UNIX trabaja bajo su propio directorio de trabajo con todas las ramificaciones conectadas, y por tanto con todas las agrupaciones lógicas de los files; si tiene necesidad de nuevos files, a veces agrupados bajo otro nombre, ocupa nuevo espacio en el disco.
Por tanto, el usuario puede proteger sus files

ALGUNOS COMANDOS DEL SISTEMA FILE UNIX

Presentado por el ordenador — **Introducido por el usuario**

```
$ pwd
/ MIO
$ ls
ABC
$ ls -a
ARCHIVO ABC PROGRAMAS
$ file ARCHIVO ABC PROGRAMAS
ARCHIVO:     directory
ABC:         ascii text
PROGRAMAS: directory
$ cd PROGRAMAS/FORTRAN; ls-a
PROG3 PROG4
$ cat PROG3
             (lista del programa PROG3)
$ mkdir SUBR
$ ls-a
PROG3 PROG4 SUBR
$ file SUBR
SUBR:          directory
$
```



Marka/Photri

Sistema de tratamiento de textos con impresora incorporada.

contra el borrado o las alteraciones efectuadas por otros usuarios; esto es posible utilizando siempre los comandos adecuados del sistema operativo. Las protecciones son de dos tipos: en escritura o en lectura. Si un file está protegido en escritura, sólo el que ha creado el file (o sea el usuario al que está asociado el directorio que lo contiene) puede borrarlo o modificar su contenido. Otro usuario que tenga otros directorios, y que se ha posicionado en el directorio examinado usando el comando cd, sólo podrá examinar su contenido. Si un file está protegido en lectura, no podrá ser leído por otros usuarios que no conozcan la clave de lectura. En el UNIX existen dos tipos de usuarios: **usuario** y **superusuario**; los primeros deben moverse en el interior de los directorios asociados a ellos, y si apuntan a otros directorios sólo pueden efectuar operaciones permitidas por los creadores (por ejemplo leer un file si éste está protegido sólo en escritura); los segundos pueden hacer de todo, en el sentido de que tienen la posibilidad de variar los mecanismos de protección en los files y en los directorios. Por norma, el superusuario es una persona que configura el sistema operativo e indica a grandes rasgos las posibilidades permitidas a los usuarios normales.

Procesos concurrentes

En los sistemas operativos que trabajan en multiusuario nace automáticamente la necesidad de coordinar las operaciones lanzadas por varios programas simultáneamente. En la multiprogramación y en el tiempo compartido ya se definen metodologías que permiten establecer qué programa debe disponer del servicio de la CPU y qué otros deben esperar su turno. Sin embargo, todavía no se han ilustrado las técnicas que permiten a estos programas intercambiar sus informaciones, o bien interrumpir la ejecución cuando el recurso que intentan utilizar ya es empleado por otro programa, y así sucesivamente. La generalización de las operaciones a coordinar se efectúa introduciendo el concepto de **proceso**: por parte de un sistema operativo, un proceso es visto como una secuencia de operaciones a realizar cada vez. No siempre un proceso coincide con un programa: puede coincidir con un procedimiento, con un subprograma o con una parte de éste; al límite, un proceso puede coincidir también con una sola instrucción, según el nivel de detalle con que el sistema operativo desglosa las operaciones que contiene el programa.

La tendencia del proyectista de un sistema operativo es casi siempre la de descomponer las operaciones en otras más sencillas hasta conseguir un nivel mínimo que coincide con una instrucción de máquina. La arquitectura determinada por este enfoque es del tipo de capas, y muy a menudo es jerárquica. El sistema operativo generado contiene N módulos enlazados entre ellos, cada uno de los cuales es responsable de controlar las operaciones que se refieren a ellos. El objetivo último de estas operaciones continúa siendo el de gestionar los recursos que se tienen a disposición, permitiendo su empleo a los usuarios que los piden.

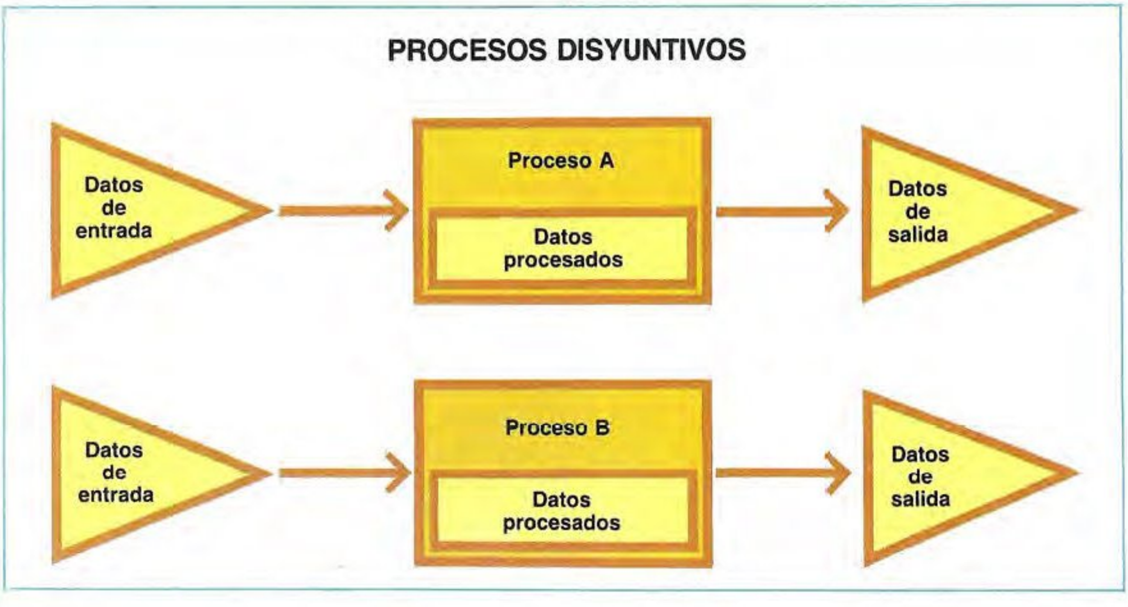
El enlace entre proceso y operaciones a efectuar está ligado al sistema operativo y a los objetivos que se prefijan. En cualquier caso, una característica común de los sistemas operativos es la de asociar a los procesos un estado o poder identificar cuál es la situación de un proceso en un momento dado.

En la multiprogramación ya se ha visto que un programa puede ponerse en ejecución, puede suspenderse por otro programa con prioridad más elevada y después reemprenderse, y así sucesivamente. En este ejemplo ya pueden identificarse los dos estados principales de un proceso: **waiting** y **running**. Cuando un proceso está en waiting está esperando la disponibilidad de un recurso para poder continuar, y cuando está en running está usando la CPU.

En los sistemas operativos existen diferentes estados de waiting: un proceso puede esperar que se haya completado una operación de entrada/salida, o bien puede esperar que otro proceso libere el recurso CPU.

En cambio, el estado de running no tiene subdivisiones y, por tanto, la tarea principal de un sistema operativo es coordinar las esperas de los procesos definiendo cuáles pueden ser las posibles transiciones entre un tipo de espera y otro, o bien entre un estado de espera y un estado de running. Las transiciones de estado de los procesos son controladas por el sistema operativo en base a los tipos de proceso en acción. Esencialmente puede haber dos tipos de procesos concurrentes: **procesos disyuntivos** y **procesos interactivos**. Dos procesos se definen disyuntivos cuando no trabajan sobre la misma base de datos, e interactivos cuando existen variables en común entre sí. Dos procesos disyuntivos tienen un comportamiento temporal independiente, en el sentido de que los resultados procesados por el primer proceso no deben ser utilizados por el segundo y, por tanto, el sistema operativo sólo debe proporcionar el servicio a uno o a otro (ver la figura de abajo).

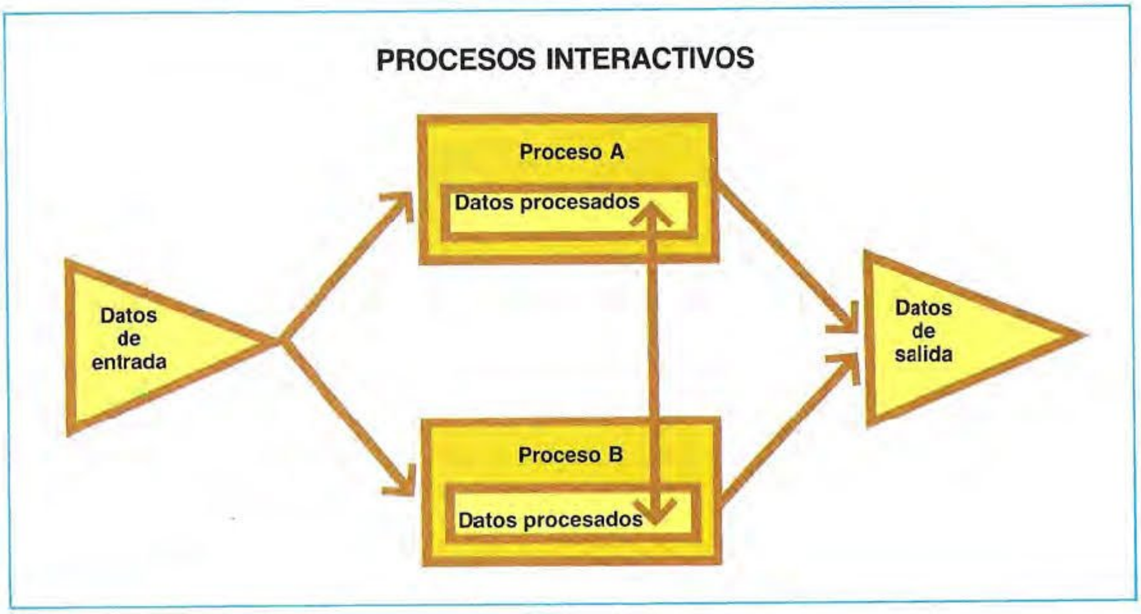
En cambio, dos procesos interactivos tienen un comportamiento temporal no previsible a priori si no se establecen las modalidades con que pueden acceder a los datos en común. Como puede observarse en la figura de arriba de la página siguiente, los datos en común también pueden procesarse mientras se ejecutan los procesos, por lo que son necesarios mecanismos de sincronización entre dichos procesos.

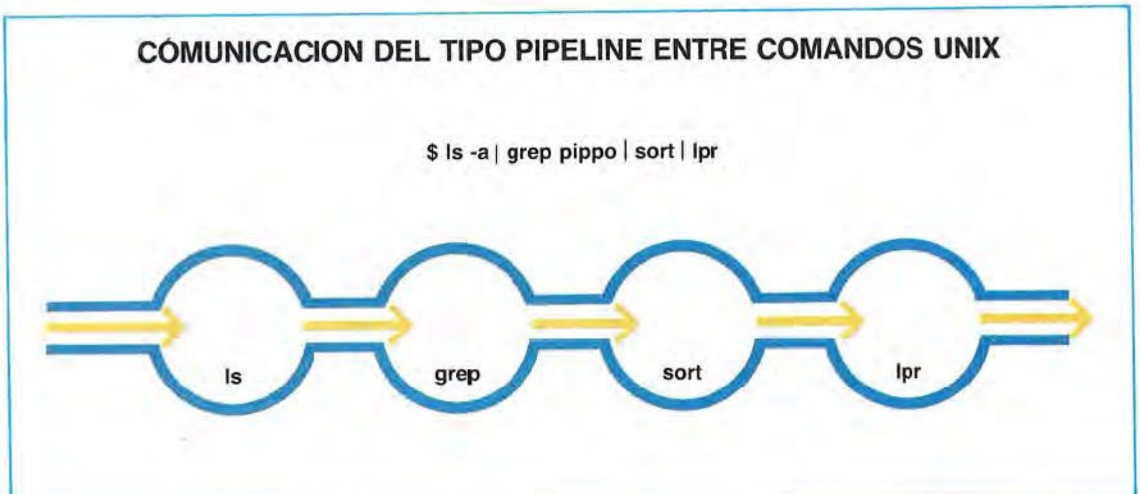


Comunicaciones entre procesos

La forma más sencilla de comunicación entre procesos es la de tipo histórico, que se tiene cuando los datos producidos en la salida de un proceso pueden usarse como datos de entrada al proceso siguiente. En el UNIX es posible el uso de esa comunicación con los mismos comandos del sistema operativo, o sea es posible establecer que los datos en la salida de un comando sean los datos de entrada de otro comando. Los dos comandos se introducen entonces secuencialmente, separados por una barra vertical que tiene el significado de concatenación o de intercambio de datos en modalidad «pipeline». En la figura de abajo se ha esquematizado la comunicación pipeline entre cuatro comandos. Con el primero (ls-a) se genera la lista de los files y directores contenidos en el directorio de trabajo (working directory), con el segundo (grep pippo) se extraen todas las líneas que contienen la palabra pippo, con el tercero (sort) se reordenan estas líneas en orden alfabético y con el cuarto (lpr) se imprimen las líneas reordenadas en una impresora. Con la comunicación histórica ya se ha preestablecido el comportamiento temporal de los procesos; cuando no es posible establecer previamente este comportamiento, tenerse en cuenta que pueden presentarse dos problemas:

- acceso a los datos compartidos
- intercambio de datos entre procesos

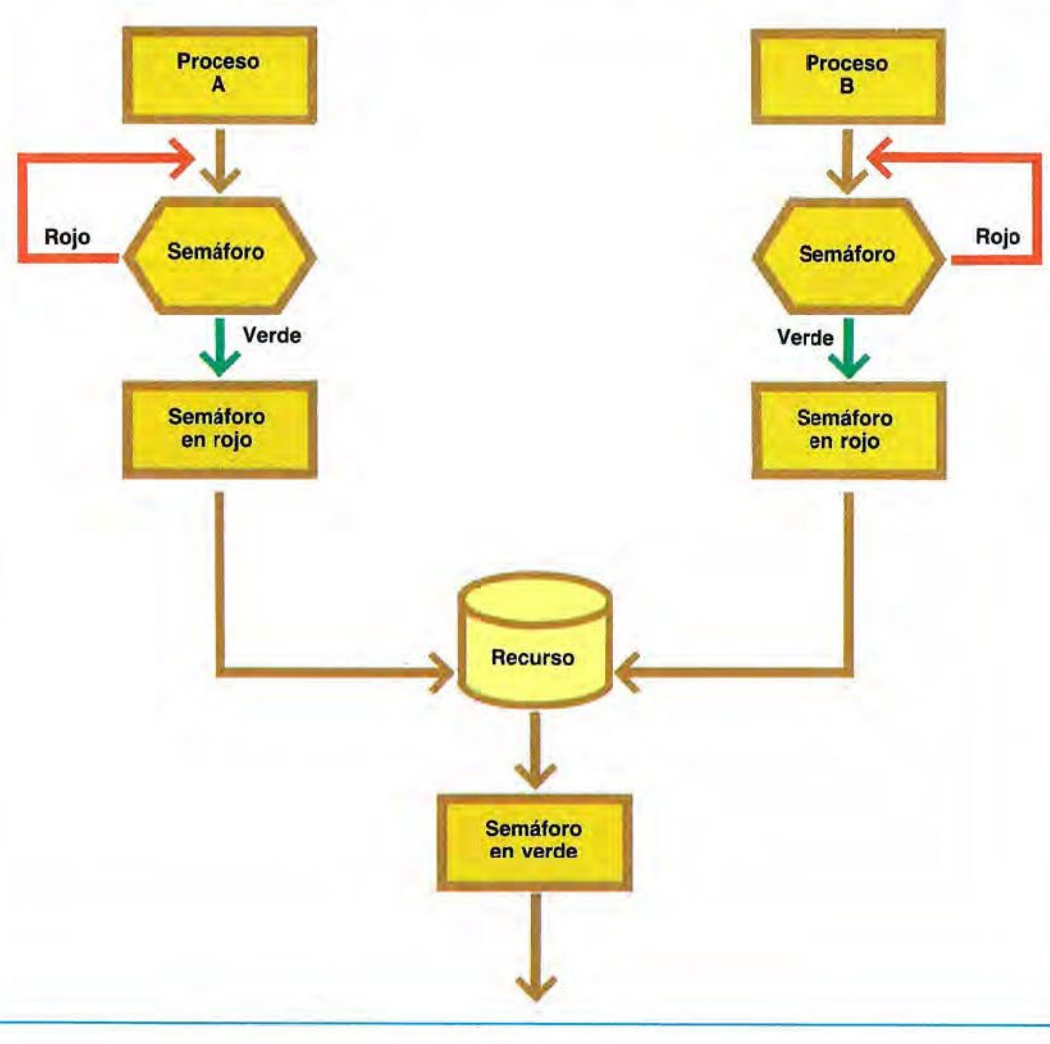




El acceso a los datos compartidos debe prever la posibilidad de que éstos puedan modificarse por el proceso que ha obtenido el acceso. En este caso deberá inhibirse el acceso a otros procesos hasta que no se hayan terminado las operaciones del que utiliza los datos.
La coordinación de estas situaciones se efectúa empleando «semáforos» puestos a disposición por el sistema operativo. El proceso que desea acceder a los datos pregunta al sistema operativo si el semáforo está en verde, o sea si se han terminado las operaciones anteriores y, por tanto, si los datos son válidos (ver la figura de abajo). Si el semáforo está en rojo, el proceso que necesita los datos tendrá que esperar a que se ponga verde antes de poder continuar.

El uso de los semáforos también vale para los recursos que no son datos; por ejemplo, es necesario evitar que las salidas de procesos concurrentes en una misma impresora se mezclen. En tal caso, el estado rojo o verde del semáforo está asociado al recurso impresora y los procesos que desean usar el recurso deben preguntar antes de su uso interrogando los semáforos gestionados por el sistema operativo. A veces pueden producirse situaciones definidas de «deadlock» en las que dos o más procesos esperan hasta el infinito que se produzca un suceso que no se produce nunca. Por ejemplo, un proceso A accede a un file en el disco y el sistema pone en rojo el semáforo conectado a éste. A continuación, el mismo proceso pide al siste-

ACCESO A UN RECURSO (FILE EN DISCO) CON USO DE SEMAFOROS



ma operativo que ponga en rojo también el semáforo conectado a una cinta magnética. Al mismo tiempo que el proceso A ha empleado el file en el disco, un proceso B ha empleado la cinta magnética e intenta poner en rojo el semáforo conectado con el file en el disco. El proceso A espera que B desbloquee la cinta y B espera que A desbloquee el file en el disco; los dos permanecen en esta situación hasta el inifinito, o hasta que un operador no interrumpe por fuerza uno de los dos procesos. Para evitar las situaciones de deadlock se definen en los sistemas operativos las llamadas zonas críticas en las cuales se gestionan los procesos concurrentes.
La gestión de los procesos en zonas críticas se efectúa en la base de tres asunciones formuladas en 1965 por Dijkstra:

1 / Cuando un proceso desea entrar en una zona crítica, la petición sólo puede durar un tiempo finito
2 / Una zona crítica sólo puede ser ocupada por un solo proceso al mismo tiempo
3 / Un proceso que se encuentre en una zona crítica sólo puede permanecer en ella un tiempo finito

El respeto de las tres asunciones de Dijkstra permite evitar las situaciones de deadlock.
El segundo aspecto inherente a la gestión de los procesos concurrentes corresponde al intercambio de datos entre los mismos, pensando en los datos como de entrada, de salida o procesador. El intercambio se produce siempre con el uso de zonas comunes a los diversos procesos en las que se depositan y de las que se toman los datos. Estas zonas pueden ser files en disco o bien zonas de la memoria central del calculador puestas a disposición por el sistema operativo para los procesos que tienen necesidad de su empleo.
El sincronismo sobre el acceso a los datos compartidos ya se ha resuelto con semáforos; sólo debe considerarse el aspecto de la comunicación. Dada una zona común, existen procesos «productores» que introducen datos en ella y procesos «consumidores» que toman datos de ella. La zona común siempre es de dimensiones finitas, por lo que en seguida aparecen problemas asociados a las diferentes velocidades con que se produce y con que se consume: si el productor es más rápido que el consumidor, la zona se llena, y no pueden introducirse otros datos en común; si el consumidor es más rápido que el productor, pueden establecerse colas en las que N procesos consumidores quedan en espera de recibir datos y, por tanto, se hace un uso no óptimo de la CPU.
En cualquier caso, las reglas que se fijan son las siguientes:

1 / Si el productor intenta introducir datos en una zona común llena, el proceso se suspende hasta que la zona común no queda aligerada de algunos datos por un consumidor
2 / Si el consumidor intenta tomar datos de una zona común vacía, el proceso se suspende hasta que un productor no haya depositado datos

Normalmente, los datos residentes en la zona común son gestionados en modalidad FIFO (First In/First Out) o LIFO (Last In/First Out). En el primer caso, los datos introducidos en primer lugar son los primeros en ser tomados; la modalidad empleada está ligada a la aplicación.
Cuando los procesos que desean comunicar son N y no dos, la zona en común debe poder distinguir los datos introducidos, por ejemplo, por el proceso A y destinados a los procesos B y C y no a otros. El sistema operativo debe utilizar mecanismos internos que le permiten tener conocimiento de los introductores y de los destinatarios de los datos. Por tanto, entre los resultados que debe garantizar está el de hacer que los procesos no destinatarios de los datos no pueden acceder a datos que no les correspondan. El resultado se consigue dividiendo en buffers la zona común y asociando a cada uno de ellos un código que sólo permite la lectura a los procesos que lo conocen. La comunicación entre procesos concurrentes puede asociarse también a usuarios concurrentes, y el intercambio de mensajes entre éstos da lugar a un servicio de «correo electrónico», del cual pueden servirse todos los que tengan un terminal conectado a un ordenador o a una red de ordenadores. El sistema operativo UNIX permite que un usuario envíe correo a todos los demás usuarios del sistema. Los destinatarios del correo, a su vez, pueden conservar los mensajes recibidos en archivos privados, formando agrupaciones sobre una base lógica (de contenido).
Finalmente es posible enviar mensajes en una base temporal; decidir que el mensaje salga en una determinada fecha o a una cierta hora, y si el destinatario coincide con el remitente, se produce una agenda electrónica para el usuario.

Los gráficos de ordenador en Basic

El sistema gráfico de ordenador constituye la aplicación más espectacular de la informática. Nacido y desarrollado originalmente sólo en grandes sistemas, recientemente ha tenido una notable difusión también en los microordenadores y ordenadores personales, difusión que ha sido posible con el aumento del potencial del hardware. Los primeros microordenadores y ordenadores personales eran máquinas limitadas a aplicaciones sencillas. La mayoría de ellas no disponía de posibilidades gráficas, y las pocas que había tenían una escasa resolución, no adecuada a los empleos profesionales.

La evolución del hardware y la disminución de los costos han permitido la construcción de sistemas que pertenecen a la familia de los micros y de los personales, pero con posibilidades gráficas ampliadas.

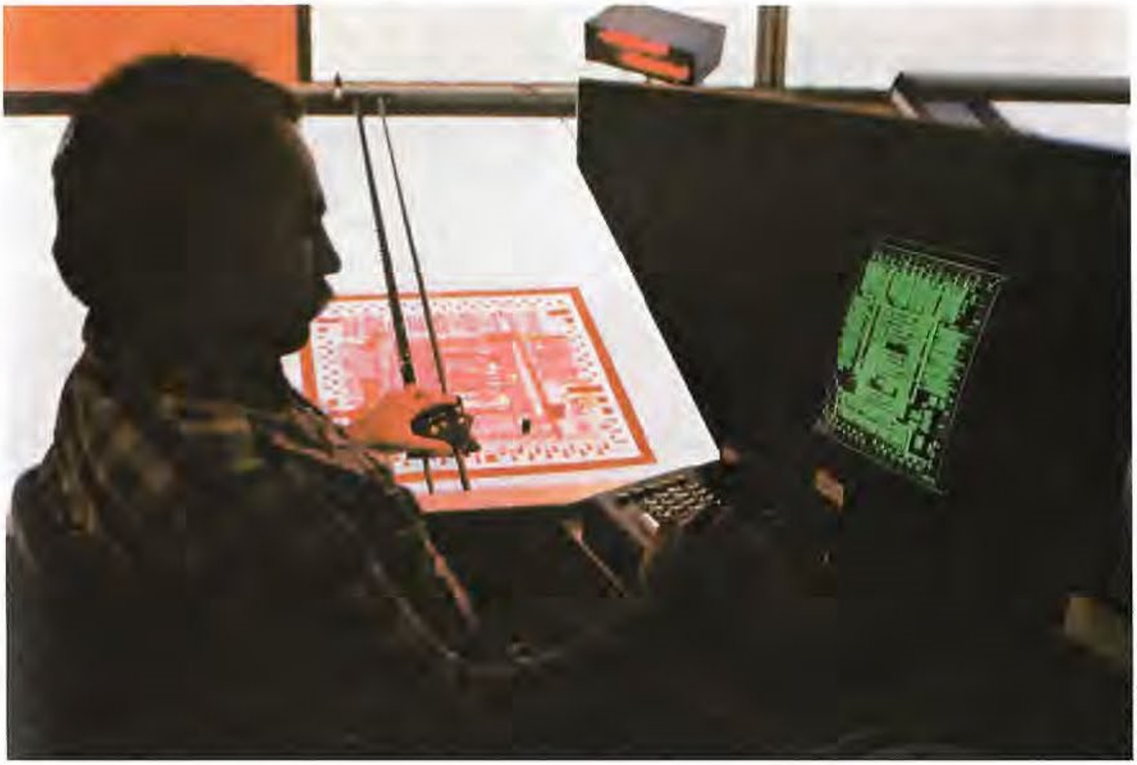
Un ordenador destinado a los usos gráficos presenta notables diferencias con respecto a los de empleos generales.

La primera diferencia corresponde a la variedad de los periféricos que debe gestionar.

En una máquina general de la categoría micro, los periféricos son esencialmente cuatro: la pantalla, el teclado, el disco (o la cinta) y la impresora. Para poder ser empleada en el proceso gráfico, la máquina debe prever muchos otros, como el plotter, la mesa gráfica y sobre todo una unidad de vídeo caracterizada por una gestión diferente y más compleja. Estos nuevos periféricos utilizan puertas de I/O y protocolos de comunicación similares a los otros, pero necesitan un software de aplicación especializado.

La segunda diferencia principal reside en la estructura hardware interna. El proceso gráfico requiere el empleo de grandes cantidades de memoria, orientando así al usuario hacia el empleo de procesadores de 16 o 32 bits o, como alternativa, al uso de fichas adicionales particulares. La primera solución, salvo raras excepciones, obliga a adoptar máquinas de la categoría mini,

Sistema de proceso para el proyecto asistido por ordenador.



Marka/J. Kramer

mientras que la segunda es la más difundida en los micros. El empleo de dispositivos adicionales está destinado a desaparecer con la aparición de los micros de 16 bits.

Aplicaciones del proceso gráfico

En la práctica puede afirmarse que no existen aplicaciones en que el proceso gráfico no pueda servir como valiosa ayuda, o cuando menos como complemento.
Para interpretar o asimilar el resultado de un proceso cualquiera, si está presentado en forma numérica, se requiere un notable esfuerzo de concentración y de análisis; si el mismo resultado se representa también gráficamente, su comprensión e interpretación son mucho más rápidas y directas.
Estas consideraciones han conducido a la aplicación del proceso gráfico también en sectores no técnicos, como por ejemplo en las investigaciones demográficas, en el análisis estadístico y en las aplicaciones económicas en general.
En función del tipo de aplicación, la gestión gráfica puede dividirse en las siguientes cuatro categorías principales:

- gráficos de gestión
- proyecto
- proceso de imágenes (image processing)
- animación

Gráficos de gestión

Con este término se indican las aplicaciones de gráficos de ordenador orientadas a los problemas de carácter económico o, más en general, al campo de las decisiones.
La automatización del trabajo de oficina en general requiere la posibilidad de representar sistemáticamente y de poner en evidencia algunos resultados importantes, y el método más inmediato para conseguir estos objetivos es precisamente la representación gráfica. El uso de los gráficos para mostrar los procesos de los fenómenos económicos ha asumido una importancia y una difusión tan amplias que ha generado un verdadero sector de los gráficos de ordenador: la gestión gráfica. Desde el punto de vista de la programación, estas aplicaciones son las más sencillas, puesto que no requieren normalmente procesos particulares o algoritmos demasiado complejos.

Proyecto

El del proyecto ha sido uno de los primeros campos de aplicación de los gráficos de ordenador en que se han aprovechado a fondo las posibilidades del calculador.
Grandes cantidades de datos correspondientes al trabajo de proyecto pueden memorizarse y reclamarse de uno o varios menús. De esta manera pueden crearse bancos de datos que contienen símbolos o detalles de un diseño ya terminado. Para obtener un diseño terminado, el proyectista sólo debe conectar los diversos símbolos, mientras que el ordenador se encarga de tener en cuenta usos particulares y, por tanto, puede proporcionar, momento a momento, la lista de los componentes insertados en el proyecto, cualquiera que sea el tema.
En las aplicaciones más sofisticadas es posible obtener varias listas del objeto, de manera que pueden obtenerse dibujos ejecutivos, o se pueden desarrollar cálculos para determinar las dimensiones o las características más adecuadas a la aplicación particular. Los principales sectores de aplicación son los siguientes:

- proyecto mecánico
- proyecto electrónico
- proyecto arquitectónico
- diseño industrial

El proyecto mecánico. Es el aspecto más conocido del proyecto asistido por ordenador, y a menudo se indica con la sigla CAD (Computer Aided Design). Está soportado por un software que puede generar dibujos mecánicos por composición de elementos geométricos elementales, o utilizando símbolos predefinidos (elementos de figura).
Los elementos geométricos principales que permiten la construcción de una figura son:

— puntos
— segmentos de recta
— círculos
— líneas curvas
— tangentes
— arcos y empalmes

Utilizando una serie de comandos para seleccionar y posicionar los diferentes elementos, pueden construirse dibujos también muy com-

plejos. Normalmente, los sistemas CAD prevén la posibilidad de generar automáticamente las partes simétricas, o de ampliar un determinado detalle, ahorrando así una notable cantidad de trabajo al proyectista. El ahorro de tiempo que se obtiene con respecto al dibujo manual tradicional es notable. La primera gran ventaja está representada por la facilidad con que pueden aportarse modificaciones; un detalle puede reclamarse en memoria y modificarse o redibujarse en pocos minutos, cuando antes eran necesarias jornadas enteras de trabajo manual.
Una segunda ventaja se debe a la posibilidad de generación automática de las instrucciones de control para las máquinas-herramienta de control numérico.
En la producción en serie se intenta eliminar lo más posible la intervención humana, particularmente en las fases de construcción de los elementos mecánicos. Existen máquinas completamente automáticas (tornos, fresadoras, roscadoras, etc.) definidas como «de control numérico» que son gestionadas completamente por un procesador. Las instrucciones necesarias para el control de las máquinas pueden generarse automáticamente durante el proyecto del elemento mecánico del sistema CAD: esta aplicación es el CAM (Computer Aided Manufacturing). El proceso global de proyecto y de generación de los códigos de control toma el nombre de CAD-CAM.

El proyecto electrónico. Esta aplicación de los gráficos del ordenador interviene de manera muy similar a lo dicho a propósito del proyecto mecánico; introducida más recientemente, está destinada a tener el máximo desarrollo en el próximo futuro. Los campos de aplicación van desde el proyecto de los circuitos integrados al de las máscaras completas, con producción automática de los dibujos ejecutivos y de la lista de los componentes necesarios para la producción. También en este sector, las ventajas van más allá del simple ahorro de tiempo asociado a la ejecución de la parte gráfica. A medida que avanza el proyecto, el uso del ordenador permite verificar la aportación de los componentes, controlar las elecciones del proyectista en términos de compatibilidad y de adaptación entre los componentes empleados y, finalmente, realizar una selección entre los diversos componentes similares sobre la base de las características requeridas.



Aplicaciones de gráficos de ordenador.

El proyecto arquitectónico. También en el campo del proyecto arquitectónico, los gráficos de ordenador se han revelado extremadamente útiles. Las principales aplicaciones corresponden al análisis de habitabilidad de los ambientes, al aparejamiento, al análisis estructural y, en los programas más complejos, al dibujo de vistas en perspectiva.
Esta última aplicación todavía está en fase de desarrollo, puesto que necesita un software y un hardware muy avanzados; las principales dificultades residen en los algoritmos de borrado de las líneas no vistas, que requieren el desarrollo de una lógica muy compleja para determinar si algún elemento de la figura debe ocultar o no una línea.
En este campo de aplicación también es muy importante la técnica de sombreado. En un dibujo hecho a mano, es la sensibilidad del artista la que decide cómo deben sombrearse para que el resultado dé la sensación de tridimensionalidad. Con el ordenador, el proceso es mucho más complejo, puesto que deben describirse a la máquina todos los procesos mentales que

conducen al resultado práctico. Los algoritmos utilizados requieren máquinas muy potentes, si bien en principio son bastante sencillas.

Diseño industrial. Las posibilidades gráficas del ordenador actualmente se emplean ampliamente en el importante sector técnico-artístico que tiene el nombre de diseño industrial.

El campo de aplicación del diseño industrial es enorme, y cubre todos los sectores en los que al proyecto técnico ejecutivo de un determinado producto es necesario asociarle un estudio estético en profundidad del resultado que se desea obtener. El ordenador ofrece la oportunidad de simplificar enormemente el trabajo de síntesis de una idea artística, que tanto puede tener la forma de un útil de trabajo como la reproducción en un tejido. Los estilistas y los creadores de moda son actualmente los usuarios más cualificados de paquetes orientados a la solución rápida de los problemas gráficos asociados a estas actividades de alto contenido creativo.

Proceso de imágenes (image processing)

En el ámbito de los gráficos de ordenador, el proceso de las imágenes asume el aspecto de una verdadera disciplina en cuanto que comporta el empleo de técnicas de software y de equipos especializados. Los principales objetivos de estas técnicas son la adquisición de las imágenes, su proceso y su reconocimiento.

Una imagen en general puede presentarse bajo varias formas (fotografía, dibujo), y suele tener el carácter de «continua»; para ser introducida en el ordenador debe «digitalizarse», o sea traducirse punto por punto a una representación binaria. Esta transformación puede realizarse con diferentes medios, según la precisión deseada; el más sencillo es la mesa gráfica, mientras que los más complejos están dotados de sistemas especializados de lectura o de telecámaras que convierten la imagen a forma digital.

También el software necesario para el funcionamiento de estos equipos presenta grandes dificultades, debidas esencialmente a la lógica binaria del ordenador. Esta última característica permite al procesador reconocer sólo los dos estados ON y OFF (0, 1), mientras que la adquisición de una imagen requiere el reconocimiento de las tonalidades de color o de los tonos de gris si se trata de una imagen monocromática. Las posibles grabaciones de color tienen un número ilimitado, mientras que el ordenador, por más potente que sea, sólo puede considerar o simular un número discreto de ellas. Por tanto, los programas de aplicación deben ser los que optimicen los recursos para obtener una representación lo más similar posible al original.

La técnica utilizada consiste básicamente en reproducir las grabaciones juntando más o menos los puntos que representan la figura; si los puntos están muy juntos se tiene un sombreado muy marcado, y separándolos se obtienen efectos de difuminado. En los sistemas más evolu-

Dos ejemplos de imágenes gráficas obtenidas con ordenador.
A la izquierda la cara de Einstein, a la derecha el puente de Brooklin.



Marka/L. de Wys



Marka/R. Villa



Centro THC/Marka

Aplicación artística del proceso de imagen en un vídeo policromático.

cionados se emplean monitores de color que, con oportunas combinaciones, permiten representaciones muy detalladas.
Como consecuencia directa del desarrollo de las técnicas de adquisición y de proceso de las imágenes, actualmente las metodologías para el reconocimiento de las mismas están en un estado de desarrollo avanzado, aunque por ahora no son fácilmente accesibles. Este campo de aplicación, llamado «reconocimientos de formatos», constituye una disciplina en sí mismo, con notables implicaciones también en otros sectores, como el de la adquisición de textos para «lectura» por parte del ordenador o la clasificación de objetos partiendo de su imagen.

Animación

Aprovechando la elevada velocidad de respuesta del ordenador, es posible preparar programas que pueden animar dibujos con una resolución tan rápida que da la sensación de movimiento continuo.
Los campos de aplicación son muy variados: desde la presentación de los videojuegos o de películas cinematográficas al análisis cinemático de un fenómeno; por ejemplo, existen programas que, simulando los desplazamientos del cuerpo de un conductor en caso de accidente automovilístico, permiten decidir cuál debe ser la mejor disposición del mismo y la mejor manera de protegerlo.

Periféricos orientados a los gráficos

Los dispositivos de I/O orientados a los gráficos comprenden una notable variedad de tipos, con campos de empleos y costos muy diversificados. A continuación examinaremos de manera más detallada los periféricos que pueden adaptarse a los microordenadores y ordenadores personales. Los programas de aplicación para los gráficos son raros, costosos y sólo disponibles para las máquinas de costo más elevado.

En la mayor parte de los casos debe ser el propio usuario el que escriba el software y, por tanto, es necesario un conocimiento profundo de los periféricos y de sus modos de gestión. En cambio, en el sector de los procesadores medios y grandes existen complejos programas dedicados, y el usuario no tiene necesidad ni manera de intervenir en el software.

Dispositivos de entrada

Los datos de entrada que deben proporcionarse a un paquete gráfico están constituidos esencialmente por una serie de coordenadas que representan el dibujo o la figura a memorizar. Por tanto, los dispositivos de entrada que es necesario emplear pueden considerarse como transductores de posición.

En la figura de la página siguiente se ha representado el esquema de principio de un dispositivo general para la adquisición de una entrada gráfica. En el plano de trabajo, por ejemplo una hoja de papel milimetrado, se ha dibujado el gráfico a adquirir. Cada punto del gráfico puede referirse a un sistema de coordenadas (X, Y) por ejemplo coincidente con los lados de la hoja.

Introducir el dibujo en la máquina significa simplemente enviar, para cada punto del dibujo, las correspondientes coordenadas.

El operador desplaza sobre el dibujo un transductor de posición que genera dos señales, la primera proporcional al desplazamiento según el eje X, y la otra proporcional al desplazamiento según el eje Y. A medida que el transductor va desplazándose a lo largo del contorno de la figura, el ordenador adquiere las coordenadas correspondientes a cada punto. La última información a enviar es el estado del punto, que puede representarse con dos valores: ON = en visión, OFF = no en visión.

Por ejemplo, posicionando el transductor en correspondencia con el punto P, el ordenador adquiere las correspondientes coordenadas (X, Y); si además se activa el estado ON, la máquina memoriza este punto y, eventualmente, lo presenta en la pantalla.

El principio físico en que se basan los transductores de posición suele ser de tipo analógico; por tanto, el periférico debe contener un convertidor analógico/digital. Además, es necesario un software que pueda referir las señales a valores compatibles con las dimensiones de la pantalla.

En la figura de la página 1390 se ha indicado el esquema de principio de un transductor potenciométrico, en el que la medida de una distancia se obtiene directamente a través de una medida de resistencia. La estructura física puede imaginarse similar a la de un tecnígrafo.

A lo largo de dos resistencias perpendiculares entre sí pueden deslizarse dos cursores metálicos; cada punto del plano puede alcanzarse de un solo modo, posicionando oportunamente los dos cursores a lo largo de las dos resistencias (guía). Una vez alcanzado el punto P, los valores de resistencia* Ry y Rx que se han medido entre los puntos P,B y B A identifican unívocamente las coordenadas del punto.

Los valores de resistencia se convierten en valores numéricos y se envían al ordenador.

El proceso puede esquematizarse en tres fases principales:

1 / lectura de la posición
2 / transformación del dato en valor numérico
3 / conversión con oportunos factores de escala

Cada una de éstas introduce imprecisiones o errores que, sumados, determinan la precisión y la repetibilidad del sistema. Las dos causas de indeterminación dependen principalmente de la construcción mecánica de los transductores y de la capacidad de resolución del convertidor A/D (analógico/digital).

El único método que puede proporcionar una evaluación realmente efectiva es la realización de una prueba práctica del aparato.

El elemento más determinante (pero no el único) es el número de bits con que trabaja el convertidor. Por ejemplo, un convertidor de cuatro bits puede proporcionar en la salida un número binario formado precisamente por cuatro bits, o sea un valor comprendido entre 0 y 15 (decimal).

La mínima cantidad que puede proporcionar es 1 y, por tanto, la precisión, como orden de magnitud, es de 1/15, o sea aproximadamente del 6%. Obsérvese que la precisión sale muy rápi-

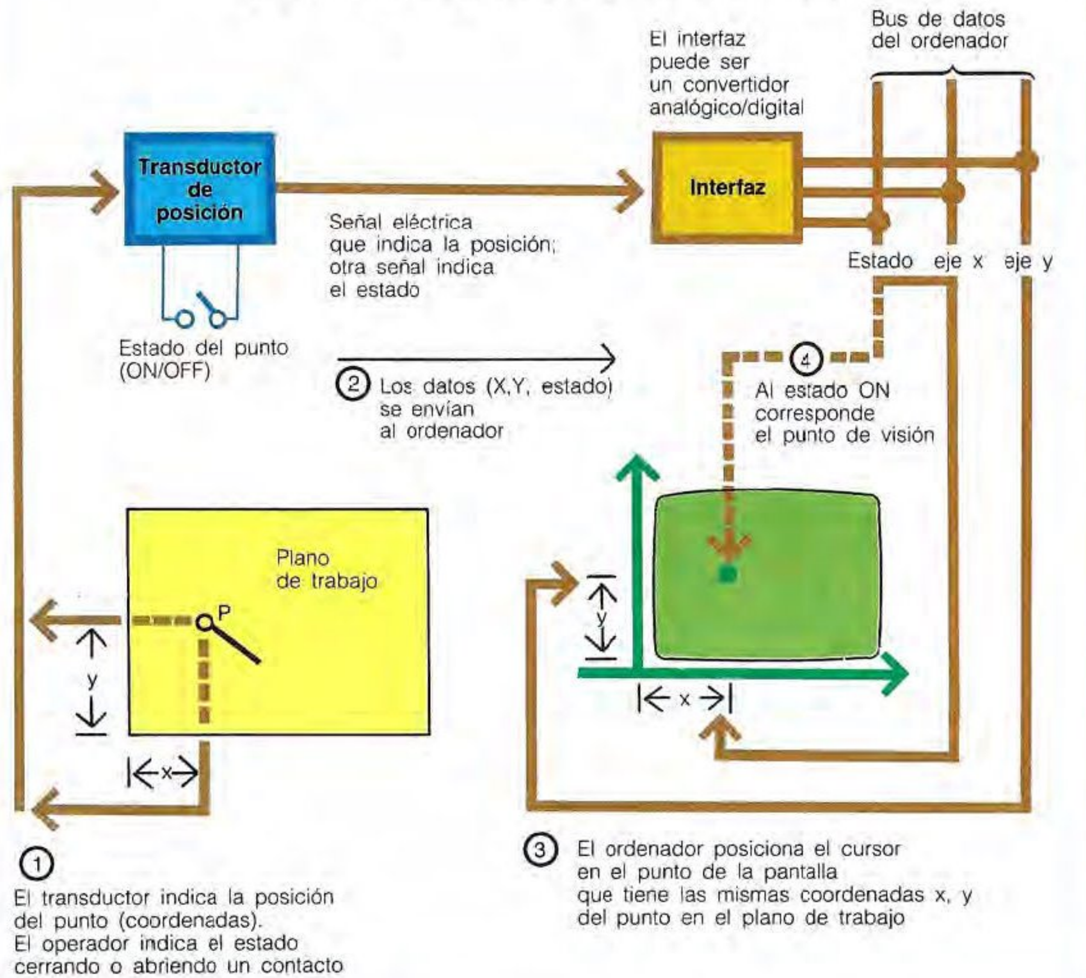
---

* El valor de una resistencia se mide en ohmios, el símbolo es Ω. Este valor depende de la longitud del hilo que constituye la resistencia (a igualdad de otras características). La dependencia se expresa por una función de proporcionalidad. Si una resistencia de 100 Ω tiene una longitud de 2 cm, posicionándose en el centro, el valor medido (entre este punto y un extremo) es de 50 Ω, si se posiciona a 1/4 es de 25 Ω y así sucesivamente. En general:

Desplazamiento = K·R

Siendo K un factor de tarado y R el valor de resistencia medido (en el ejemplo, K = 2 cm/100 Ω = 1/50 [cm/Ω]).

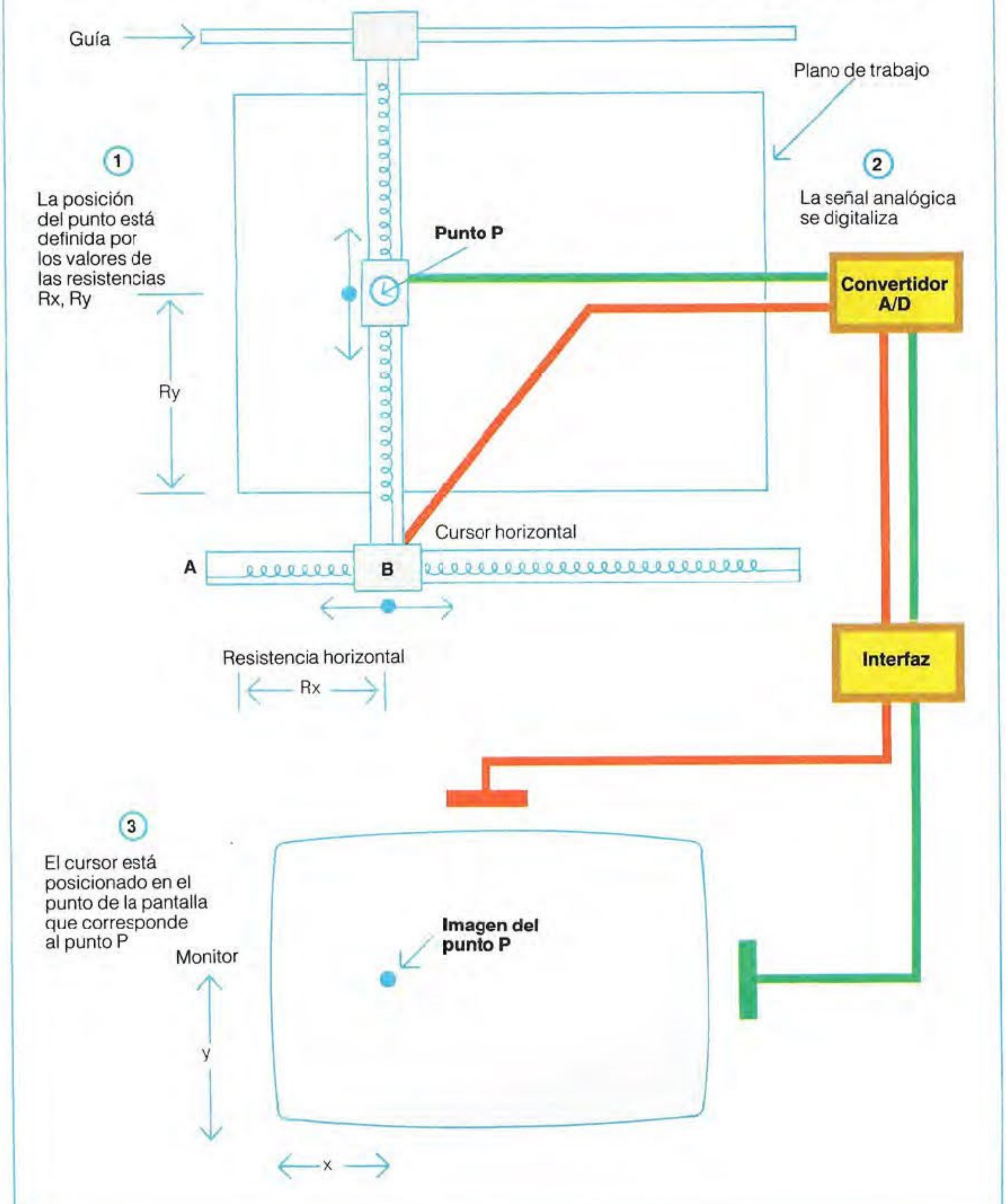
ESQUEMA DE PRINCIPIO DE UN PERIFERICO DE ENTRADA PARA EMPLEOS GRAFICOS



damente con el número de bits; por ejemplo, con un convertidor de 5 bits se hace igual a 1/31, o sea aproximadamente el 3%.

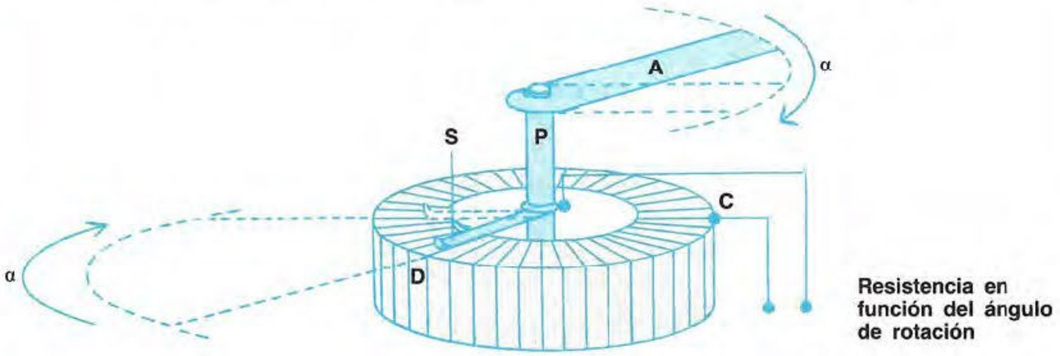
La mesa gráfica. El transductor potenciométrico es muy empleado en las mesas gráficas, con una variante respecto al esquema descrito. La construcción de los potenciómetros de forma rectilínea y de longitud igual a la de una hoja de dibujo es muy cara e implica además el uso de un sistema de guías. En cambio es mucho más económica la adopción de potenciómetros de desarrollo circular (en la figura de arriba de la página 1391 se han representado los principales tipos). Sin embargo, este transductor puede medir un ángulo y no una distancia; para obtener las coordenadas de un punto debe considerarse un sistema de referencia de **coordenadas polares** del tipo esquematizado en la figura de abajo de la página 1391. En él, las posiciones de un punto en general del plano se identifican por un ángulo y una longitud. Por ejemplo, el punto P tiene por coordenadas polares el ángulo α y el segmento A, mientras que el punto P1 se identifica por $\alpha 1$ y A1. Refiriéndonos a la figura de arriba de la página 1391, la distancia A es la longitud del brazo y el ángulo α es la rotación, de la cual depende el valor de resistencia. Girando el brazo, el valor de la resistencia (R) es proporcional al ángulo, y por tanto pueden identificarse todos los puntos que pertenecen a la circunferencia de radio A y centro en el centro

ESQUEMA DE UN TRANSDUCTOR POTENCIOMETRICO DE POSICION



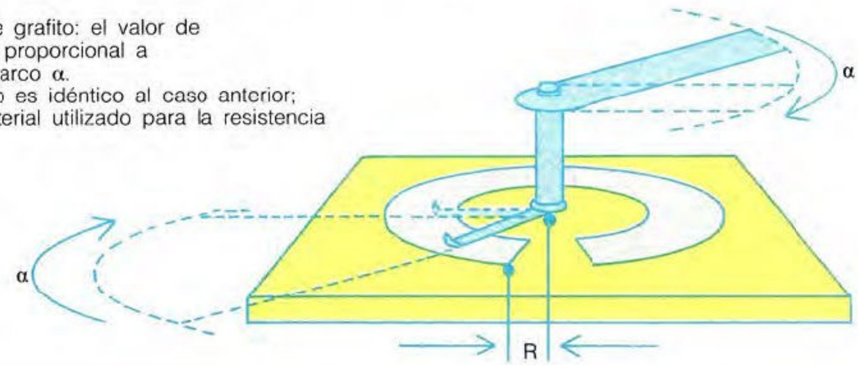
del potenciómetro. Este sistema no permite obtener la posición de todos los puntos del plano, sino sólo los que pertenecen a la circunferencia descrita por el extremo del brazo en su rotación. Para obtener todos los puntos debe montarse un segundo potenciómetro en el extremo del brazo A y un segundo brazo B. De esta manera pueden alcanzarse todas las posiciones del plano de trabajo. El esquema final se ha representado en la figura de la pág. 1392.

POTENCIOMETROS UTILIZADOS EN LA MESA GRAFICA

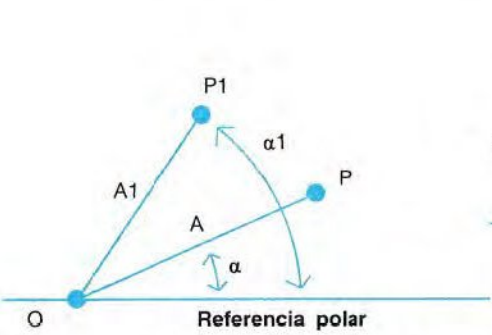


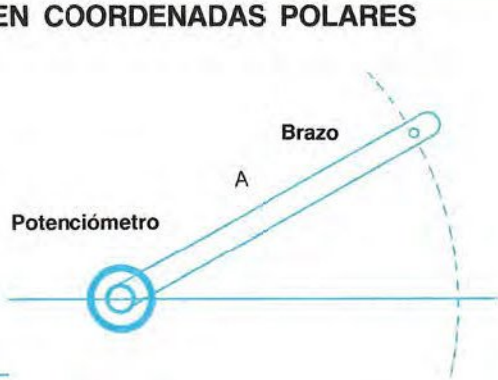
Esquema de funcionamiento de un potenciómetro. La rotación del brazo A produce la rotación del eje P que, a su vez, mueve el contacto deslizante S. Variando la posición de S varía también la cantidad de hilo eléctrico comprendido entre los puntos C y D, de los cuales el primero es un extremo del arrollamiento y el segundo es el contacto S. Para potencias limitadas, como en las utilizadas en este tipo de aplicación, el arrollamiento de hilo se sustituye por una fina capa de material adecuado (por ejemplo, grafito).

Potenciómetro de grafito: el valor de la resistencia es proporcional a la magnitud del arco α.
El funcionamiento es idéntico al caso anterior; sólo varía el material utilizado para la resistencia

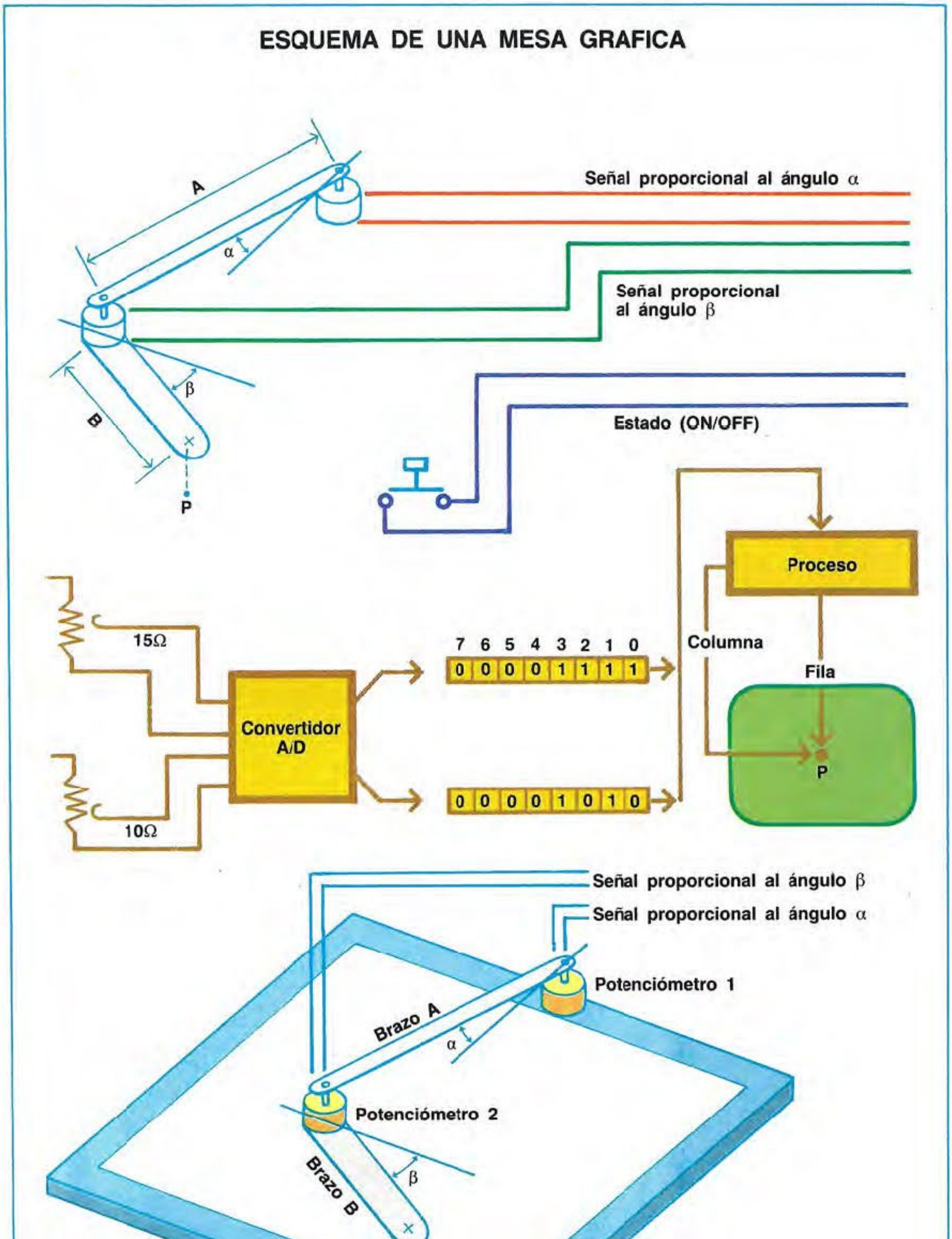


SISTEMA DE REFERENCIA EN COORDENADAS POLARES





Coordenadas polares. La posición de un punto puede identificarse con un ángulo y una distancia.
La distancia es la longitud del segmento que une el punto examinado con el origen O; el ángulo es el comprendido entre el segmento y una recta de referencia.
En el ejemplo, el punto P tiene coordenadas (A,α), mientras que el punto P1 tiene coordenadas (A1, α1).


ESQUEMA DE UNA MESA GRAFICA
A
α
B
β
P
Señal proporcional al ángulo α
Señal proporcional
al ángulo β
Estado (ON/OFF)
15Ω
10Ω
Convertidor
A/D
7 6 5 4 3 2 1 0
0 0 0 0 1 1 1 1
0 0 0 0 1 0 1 0
Proceso
Columna
Fila
P
Señal proporcional al ángulo β
Señal proporcional al ángulo α
Potenciómetro 1
Brazo A
α
Potenciómetro 2
Brazo B
β
Pulsador de selección
de estado (ON/OFF)

Las coordenadas proporcionadas por este sistema son angulares; por esto deben transformarse en cartesianas (X,Y). El método, muy sencillo, se ha representado en la figura de abajo. La fórmula resolutiva es

$$X = A * COS(\alpha) + B * COS(\beta)$$
$$Y = A * SEN(\alpha) + B * SEN(\beta)$$

habiendo indicado:

X,Y = coordenadas cartesianas del punto

A,B = longitudes de los dos brazos

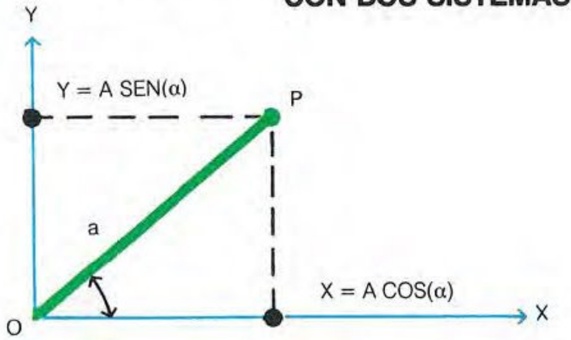
α,β = ángulos de los dos brazos con respecto a una recta de referencia

Normalmente, las longitudes de los dos brazos son iguales (A = B), por lo que

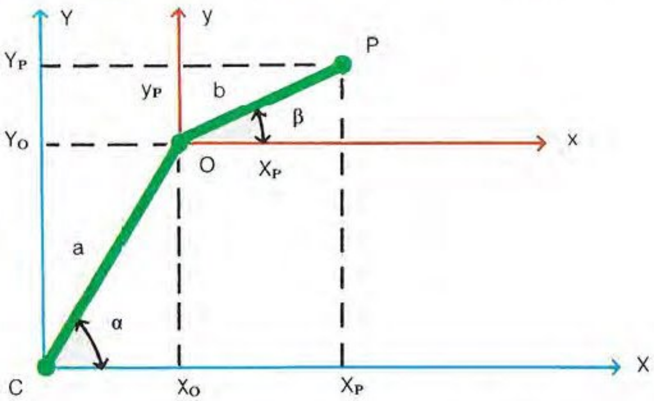
$$X = A * (COS(\alpha) + COS(\beta))$$
$$Y = A * (SEN(\alpha) + SEN(\beta))$$

Los valores α y β los proporcionan dos convertidores A/D (uno para cada potenciómetro) en

DETERMINACION DE LAS COORDENADAS DE UN PUNTO CON DOS SISTEMAS POLARES



La transformación de coordenadas polares (α, A), en coordenadas cartesianas (X,Y) es inmediata haciendo coincidir los orígenes de los dos sistemas de referencia (punto 0) y adoptando como recta de referencia de los ángulos el eje X. Las coordenadas cartesianas son las proyecciones del segmento A sobre los ejes.



Las coordenadas del punto P con respecto a los ejes x, y son

$x_P = b \cdot COS(\beta)$
$y_P = b \cdot SEN(\beta)$

Las del punto O con respecto a los ejes X, Y son

$X_O = a \cdot COS(\alpha)$
$Y_O = a \cdot SEN(\alpha)$

Las coordenadas de P referidas al sistema X, Y se obtienen sumando las coordenadas de P con respecto al sistema x, y con las de O con respecto al sistema X, Y:

$X_P = X_O + x_P = a \cdot COS(\alpha) + b \cdot COS(\beta)$
$Y_P = Y_O + y_P = a \cdot SEN(\alpha) + b \cdot SEN(\beta)$

Siendo a y b constantes (longitudes de los brazos de la mesa gráfica), la posición del punto P está determinada por los valores de los ángulos de rotación necesarios para posicionar el elemento de colimación en P.

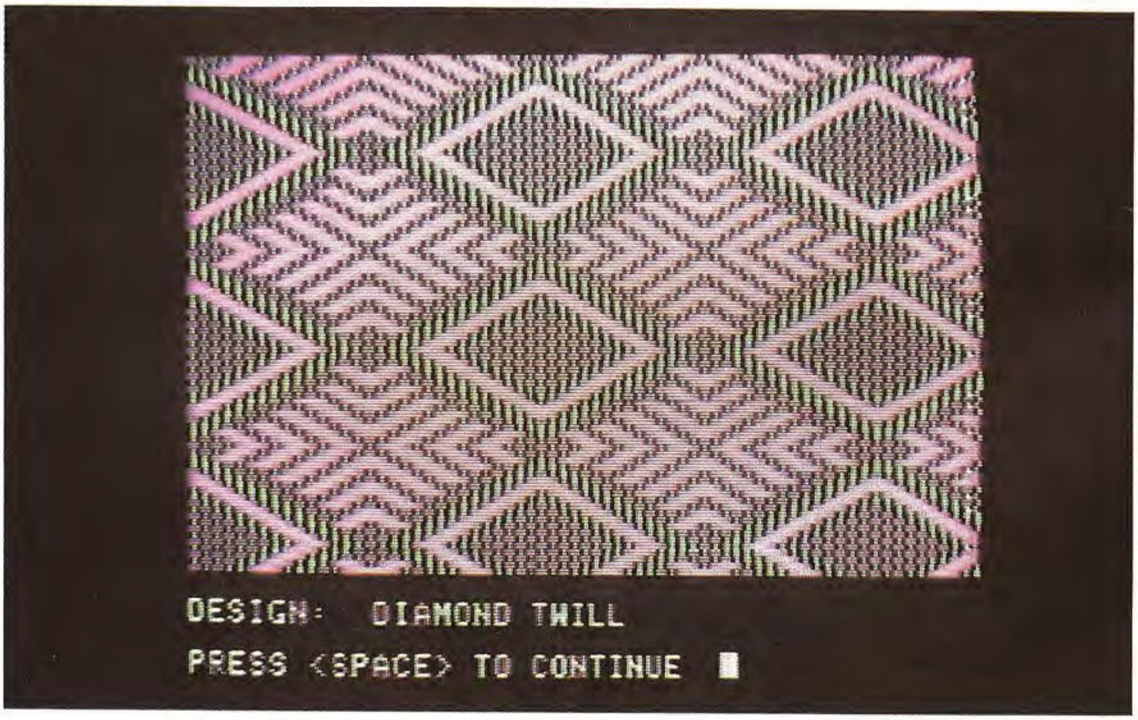
forma de valores numéricos digitales; el software debe limitarse a calcular las dos sencillas expresiones indicadas.
Los traductores potenciométricos no son los únicos utilizados. La principal limitación se debe al espacio ocupado por los brazos en el plano de trabajo y a las necesarias tolerancias mecánicas. Con los dispositivos «de pluma» se tiene una disposición mejor.
El transductor comprende como elemento sensible una pluma cuya punta, en lugar de escribir, está constituida por un interruptor que, en el momento de la presión sobre el papel, se cierra y señala el estado ON/OFF del punto. Un transductor así necesita sistemas de detección de las coordenadas mucho más complejos que el anterior. Los más usados son dos.

- Campos eléctricos o magnéticos. La mesa genera un campo eléctrico o un campo magnético; la pluma contiene un sensor que proporciona una tensión proporcional a la intensidad del campo que la envuelve y, por tanto, es una función de la posición.
- Ultrasonidos. La pluma emite continuamente ondas ultrasonoras. Unos adecuados detectores posicionados en los bordes del papel miden el tiempo que transcurre entre el instante en que se produce la descarga y el de recepción del sonido, que es proporcional a la distancia y, por tanto, permite obtener la posición de la pluma con respecto a los detectores, o sea las coordenadas.

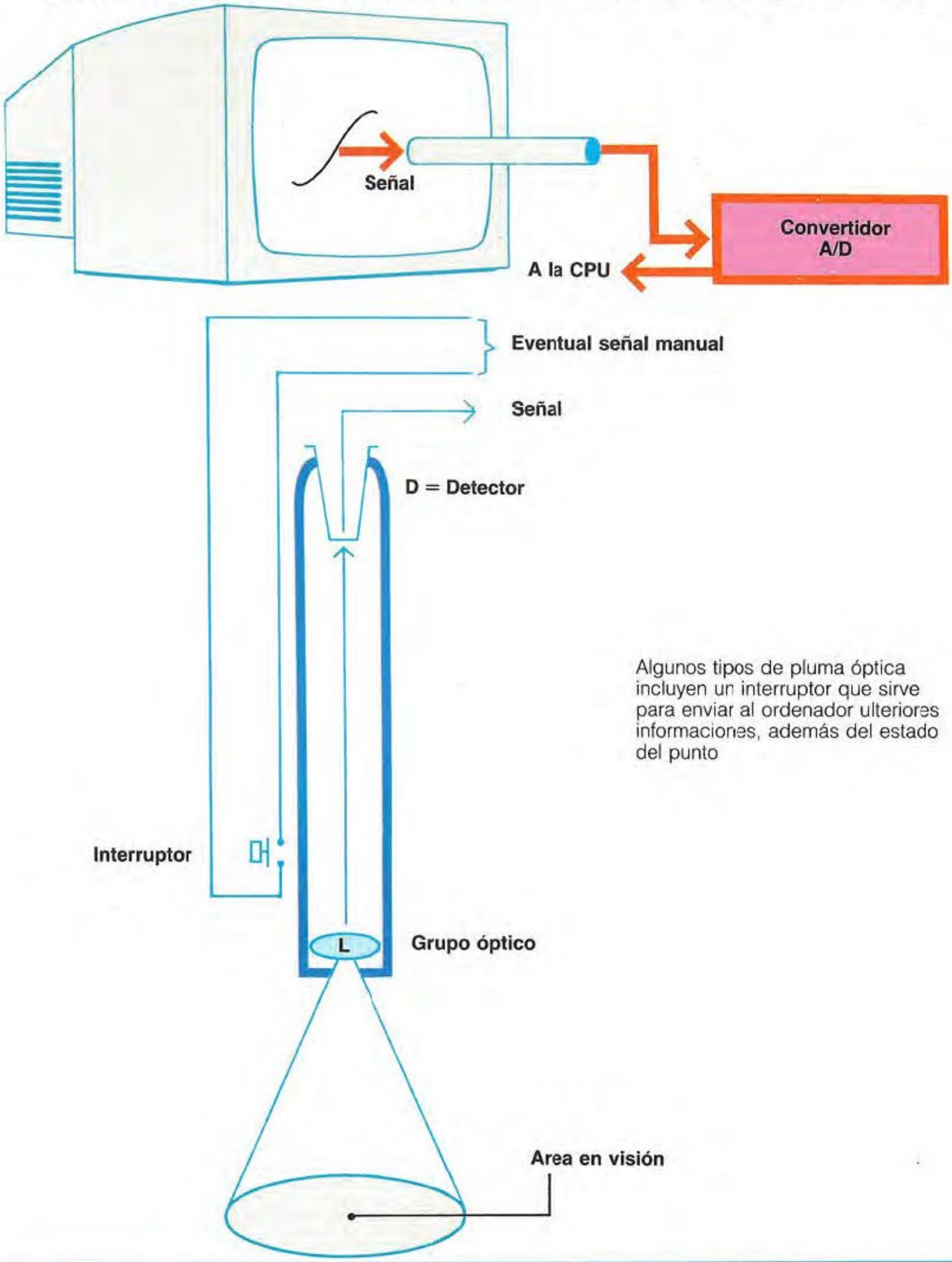
Estos métodos necesitan sistemas de hardware muy complejos, y normalmente se utilizan en las aplicaciones que corresponden a los microordenadores y a los ordenadores personales.
También existen algunos dispositivos, normalmente mecánicos, que permiten detectar las medidas de objetos tridimensionales. En estas aplicaciones, además de los importantes costos del hardware, se presentan notables complicaciones de software, puesto que deben memorizarse y procesarse figuras tridimensionales.

Dispositivos especiales. Además de los ya vistos, existen otros métodos para comunicar informaciones gráficas al ordenador, si bien más complejos y menos difundidos que los anteriores. Los dos sistemas principales, también de reciente aplicación en los microordenadores y ordenadores personales, son la pluma óptica (light pen) y la pantalla vídeo sensible al tacto. En la figura de la página siguiente se ha repre-

Empleo del proceso gráfico en el diseño del dibujo de un tejido.



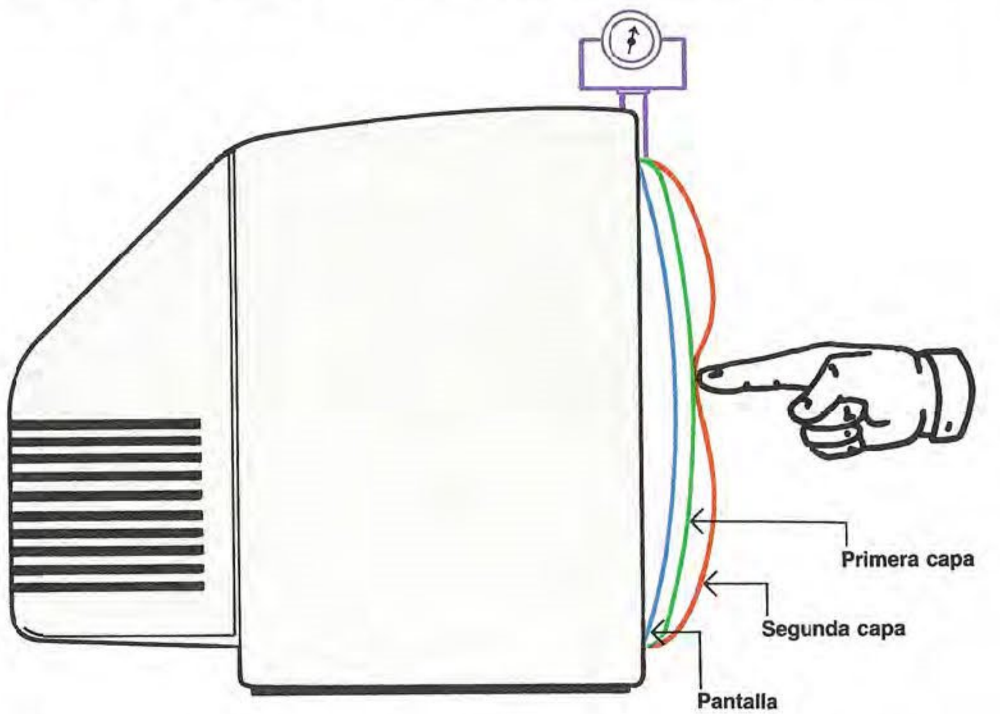
ESQUEMA DE FUNCIONAMIENTO DE LA PLUMA OPTICA (LIGHT PEN)



Algunos tipos de pluma óptica incluyen un interruptor que sirve para enviar al ordenador ulteriores informaciones, además del estado del punto

sentado el esquema de principio de una pluma óptica. Básicamente se trata de un detector sensible a la luz emitida por el vídeo. Desplazando la pluma sobre la pantalla, la luz recibida se convierte en señales eléctricas que se envían al ordenador (después de su conversión analógico/digital). Esta señal, procesada por una unidad de control conectada a la pantalla, permite obtener las coordenadas de la pluma.
Este medio permite «leer» un dibujo sobre la pantalla o crear uno como si la pluma óptica fuese un lápiz normal que discurriese por un papel.

PRINCIPIO DE FUNCIONAMIENTO DEL VIDEO SENSIBLE AL TACTO



Arriba se ha representado un esquema que ilustra el funcionamiento de un vídeo sensible al tacto. La superficie exterior de la pantalla está constituida por dos hojas de material transparente separadas por un fino espacio. Ejerciendo una ligera presión, la hoja más exterior se curva hasta tocar la más interior; de esta manera se tiene una variación de la diferencia de potencial entre las dos hojas que, convertida a forma digital, proporciona las coordenadas en que se ha realizado el contacto. Es un medio todavía impreciso si se compara con los otros, y hoy puede emplearse sólo para usos secundarios.

Dispositivos de salida

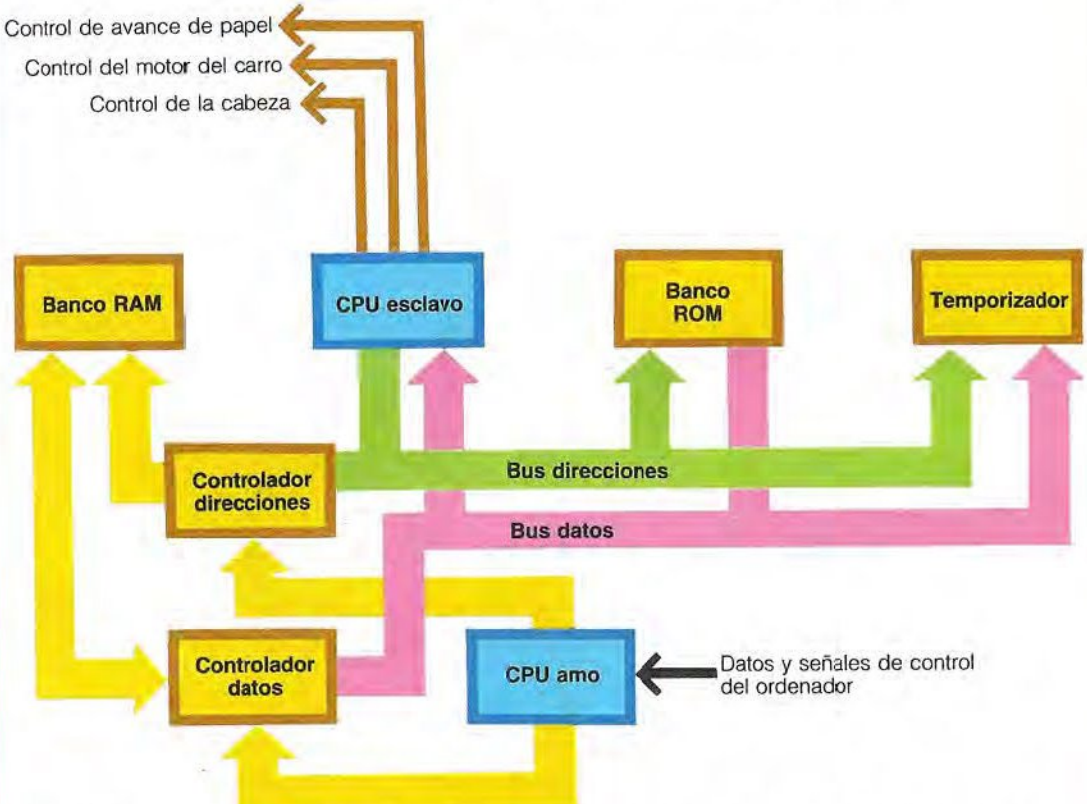
Los principales dispositivos de salida orientados a las aplicaciones gráficas son la impresora, el trazador gráfico o plotter y el vídeo gráfico.

La impresora gráfica. Para los empleos gráficos que no necesitan una elevada precisión pueden emplearse las impresoras llamadas semigráficas o gráficas.

Se trata de impresoras normales de agujas (ver la figura de la página siguiente) que sin embargo permiten direccionar cada elemento simple de la matriz. En el empleo normal (impresión de caracteres) a cada código ASCII enviado por el ordenador corresponde una determinada serie de comandos memorizados en una ROM que activan, entre todas las agujas de la matriz, sólo las correspondientes al carácter a imprimir. En el modo gráfico es posible eliminar este proceso interno de la impresora para controlar con el ordenador cada aguja simple de la matriz. Los gráficos que se obtienen no son muy precisos y su gestión es lenta, sobre todo por las limitaciones de movimiento que tiene el carro. La cabeza de escritura sólo puede moverse horizontalmente; el movimiento de la otra coordenada puede obtenerse haciendo avanzar el papel en pasos de una línea, por lo que el gráfico producido tiene, según uno de los dos ejes, una resolución que como máximo es igual a la distancia de entre líneas de la impresora.
Seleccionando oportunamente las agujas pue-

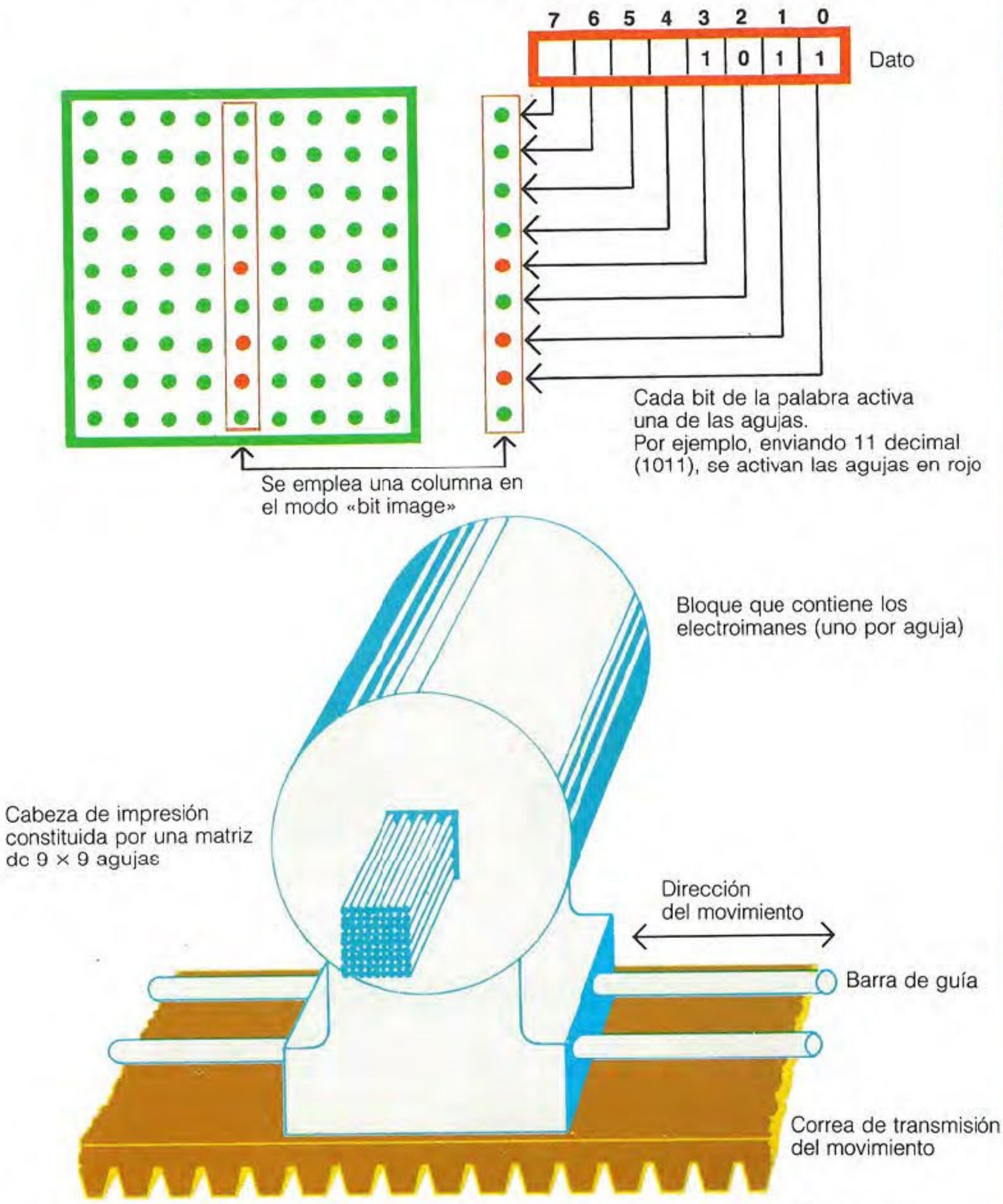
de obtenerse alguna mejora, pero continúa obteniéndose una presentación deficiente desde el punto de vista gráfico.
En la figura de la página siguiente se ha esquematizado un método que permite mejorar la calidad del gráfico. El método se llama «colum scan bit» y permite seleccionar una de las columnas que componen la cabeza de escritura y controlar cada una de las agujas de manera autónoma con respecto a las otras. Puede aplicarse el mismo método sobre una línea de la matriz (aunque no todas las impresoras lo prevén) y esto permite direccionar un determinado número de puntos (7 × 9 o 9 × 9, según las dimensiones de la matriz) en la restringida área ocupada por un carácter, con un sensible aumento de la resolución del gráfico. En algunas máquinas existen rutinas de sistema que permiten reproducir en la impresora todo el contenido de la pantalla del vídeo.
En muchas máquinas existe un área de memoria llamada página gráfica que contiene la imagen digitalizada de lo que aparece en la pantalla. Las rutinas escritas toman el contenido de esta área de memoria y, en función de las cifras binarias en ella contenida, activan las correspondientes agujas de la cabeza.
En este caso, las dificultades están en el mecanismo de direccionamiento y selección de las posiciones de memoria en el interior de la página gráfica. Además, en algunas aplicaciones, antes de la transferencia sobre el papel, debe

ESQUEMA DE UNA IMPRESORA SEMIGRAFICA



La CPU amo recibe los datos y las señales de control del ordenador (o mejor, de los circuitos de interfaz) y los transfiere a la RAM o activa una de las ROM, según el modo operativo.
Si es una escritura normal de caracteres se conecta con el bus de datos la ROM que contiene el juego de caracteres seleccionado. La CPU esclavo toma estas señales y las envía a los órganos de potencia que controlan los dispositivos mecánicos de impresión.
En el modo gráfico, los datos recibidos por la CPU amo se transfieren a la memoria RAM y de ésta, siempre a través de la CPU esclavo, a los órganos mecánicos.
En las impresoras más modernas hay diferentes ROM según el tipo de carácter que quiere obtenerse.

METODO DE IMPRESION «BIT IMAGE»

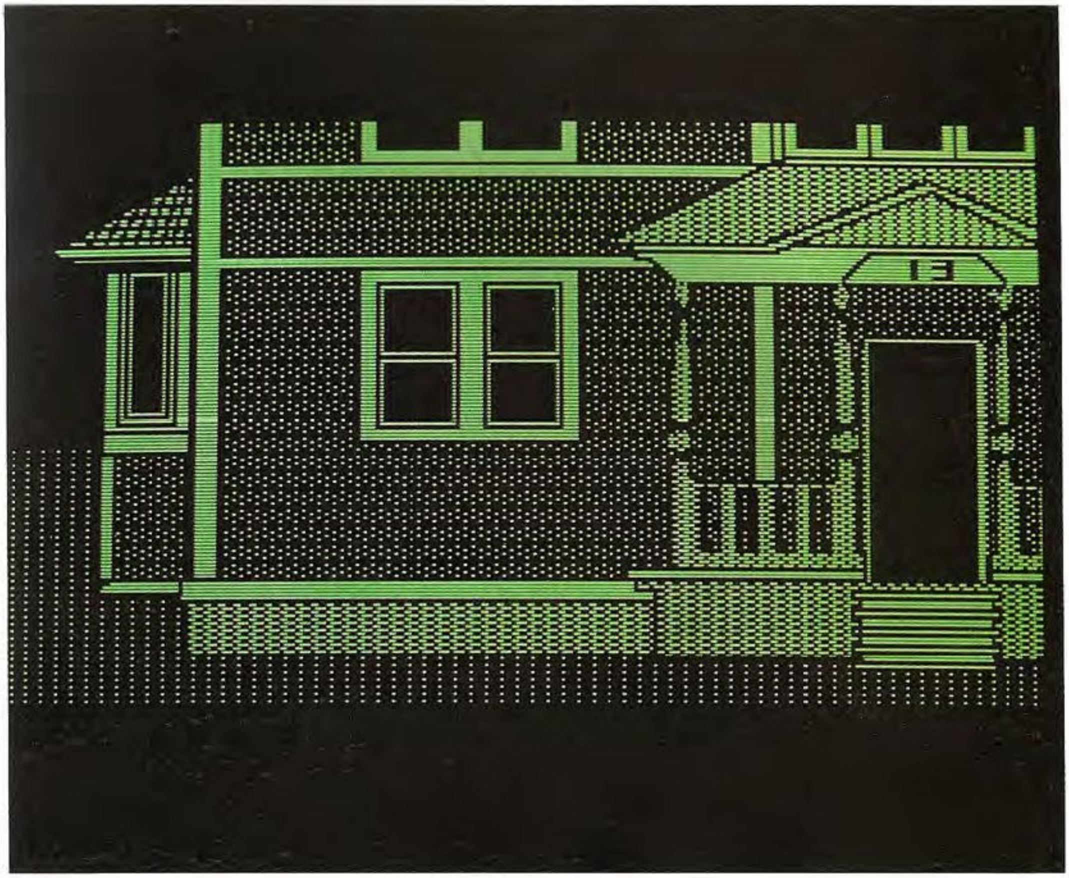


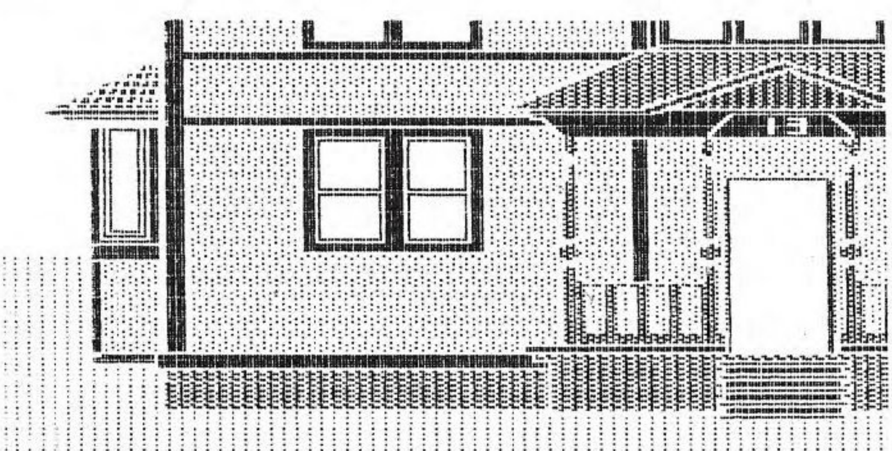
Este tipo de funcionamiento es activado normalmente en dos tiempos. Inicialmente se envía un código de control seguido de un valor numérico que expresa el número de bits a representar (que se enviarán sucesivamente). En un segundo tiempo deben enviarse los datos, agrupados en palabras de siete u ocho bits, con la lógica expuesta en el gráfico.

invertirse la pantalla del vídeo, puesto que en ella, el dibujo está trazado en blanco sobre negro (o también en el color de los fósforos sobre fondo oscuro), mientras que la impresora sólo puede escribir en negro sobre blanco; si el gráfico es un dibujo con sombreados, la inversión produce resultados desastrosos.

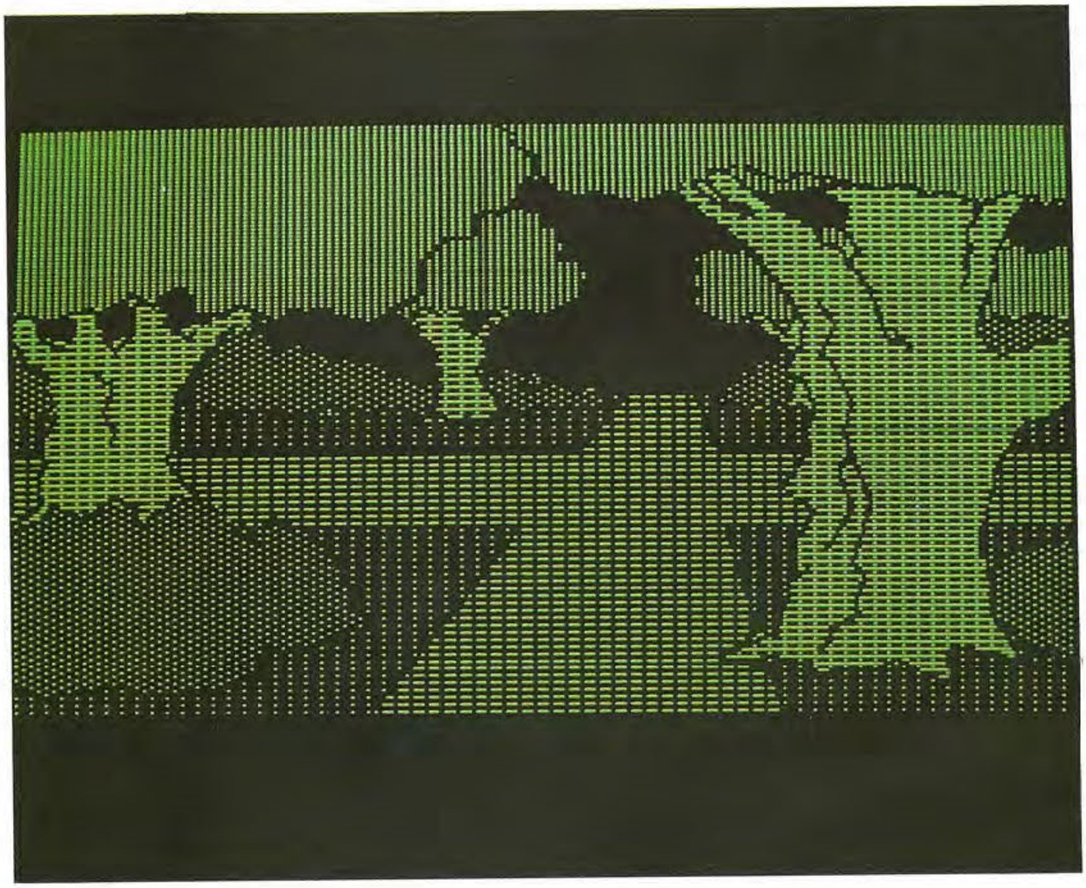
El plotter. Es la unidad de salida más empleada en las aplicaciones de carácter técnico, puesto que permite obtener gráficos de calidad similar e incluso mejor que los hechos a mano sobre el tradicional tablero.Los plotters pueden dividirse en dos categorías, según el método utilizado para trazar el dibujo:

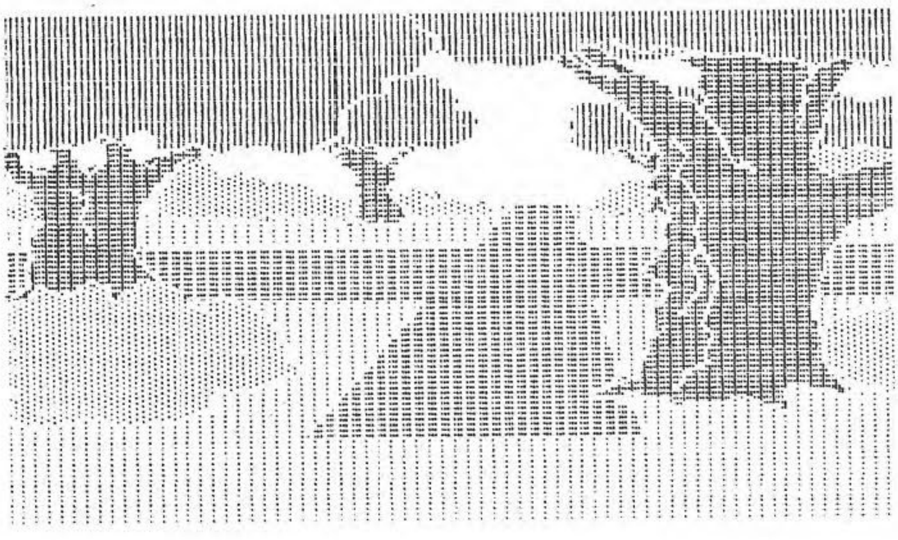
SALIDA POR IMPRESORA DE UNA IMAGEN GRAFICA (1)





SALIDA POR IMPRESORA DE UNA IMAGEN GRAFICA (2)





- plotters de pluma
- plotters electrostáticos

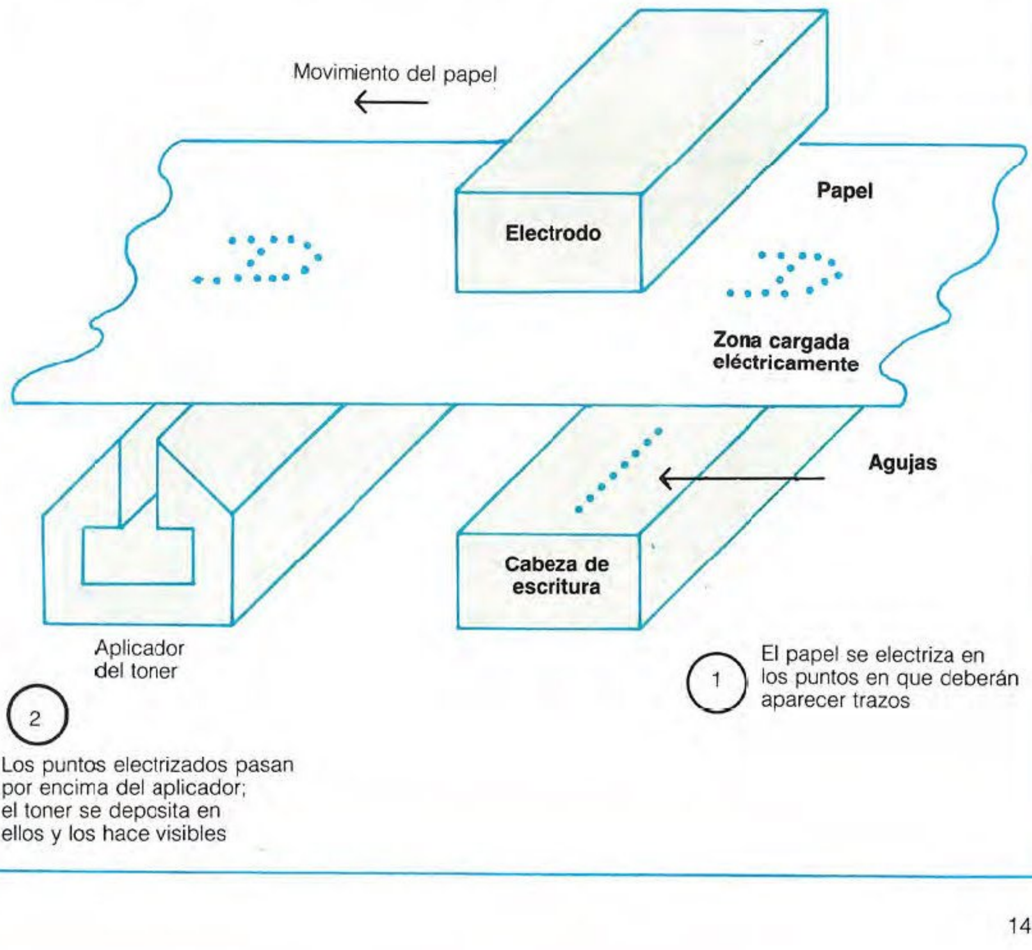
En los plotters de pluma, el elemento que traza el dibujo es una pluma muy similar a las empleadas por los dibujantes; en los electrostáticos, llamados también **rasters**, la escritura se obtiene cargando eléctricamente los puntos del papel que deben aparecer dibujados y depositando a continuación un toner sobre el papel. En la figura de abajo se ha representado el esquema adoptado en este sistema, que es idéntico al utilizado para las impresoras electrostáticas. Este tipo de periférico presenta dos desventajas: la primera es la necesidad de utilizar papel tratado y, la segunda, la imposibilidad de dibujar varias veces el mismo punto, por lo que el dibujo, antes de enviarse al plotter, debe descomponerse en líneas (de esta operación deriva la denominación de raster).

El tipo de plotter más empleado es el de pluma, porque permite asociar una calidad gráfica excelente a la facilidad de uso. La pluma puede moverse en todas direcciones y también pueden volver a pasar sobre zonas parcialmente ya dibujadas. Esta característica deja amplia libertad al usuario para preparar el software del modo que prefiera, sin los vínculos impuestos por la estructura física de la máquina.

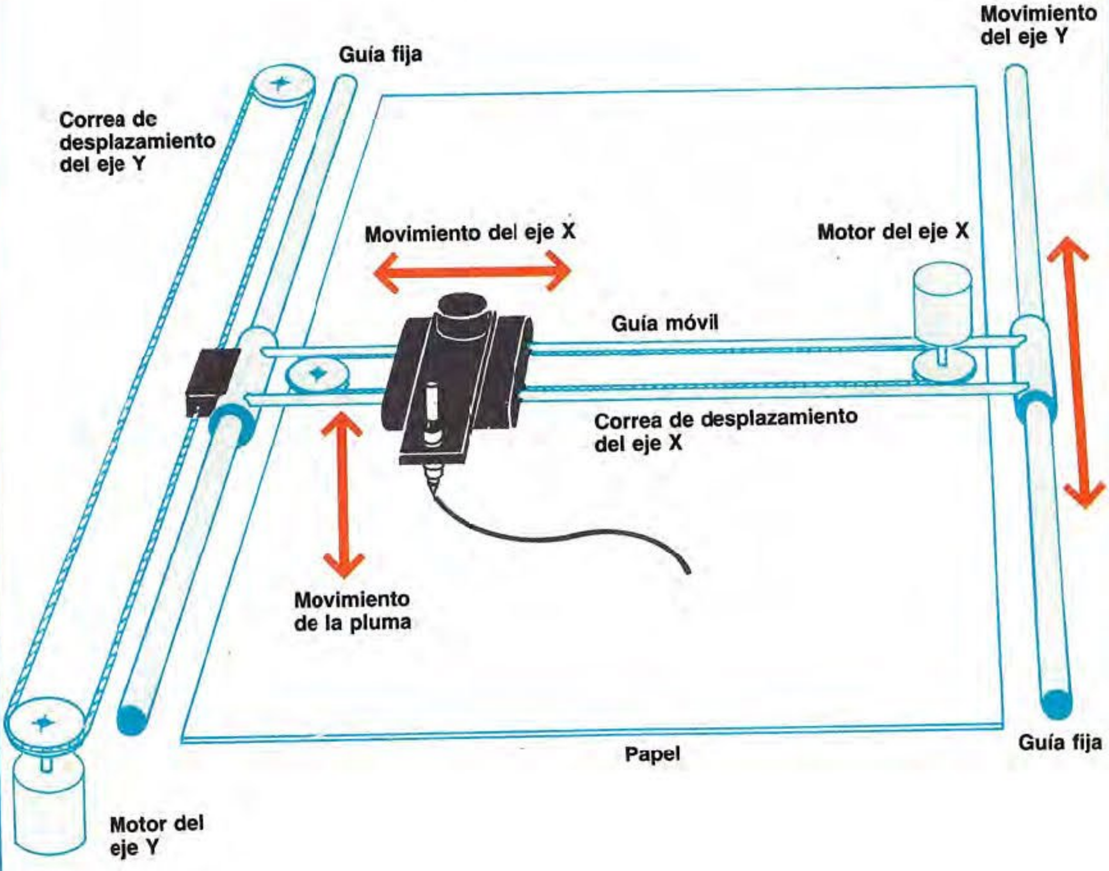
Los plotters pueden clasificarse también en base al tipo del movimiento realizado. Los tipos más difundidos son los plotters planos y los de tambor. Otros tipos, como el plotter de rodillo y el de movimiento del papel, son variantes de los dos tipos ya descritos.

El **plotter plano** está constituido por un plano que soporta la hoja de papel y por dos guías fijas paralelas, a lo largo de las cuales se desplaza una guía móvil. En la guía móvil se ha montado el carro portapluma (ver la figura de

PRINCIPIO DE FUNCIONAMIENTO DEL PLOTTER ELECTROSTATICO



ESQUEMA DE PLOTTER PLANO



arriba). Los movimientos (X,Y) necesarios para trazar el dibujo se obtienen con dos motores: el primero desplaza la guía móvil (eje Y) y el segundo desplaza el carro a lo largo de la guía móvil (eje X). Además de estos movimientos, también debe haber el que sube o baja la pluma, obtenido mediante un electroimán montado sobre el carro portapluma.

De la mecánica del periférico es evidente la extrema sencillez de su uso; sólo son necesarios dos comandos, uno para desplazar la pluma del punto general de coordenadas X,Y, y el otro para especificar si el desplazamiento debe hacerse con la pluma subida o bajada.

En el primer caso puede tenerse el trazado del segmento que une la posición actual con el

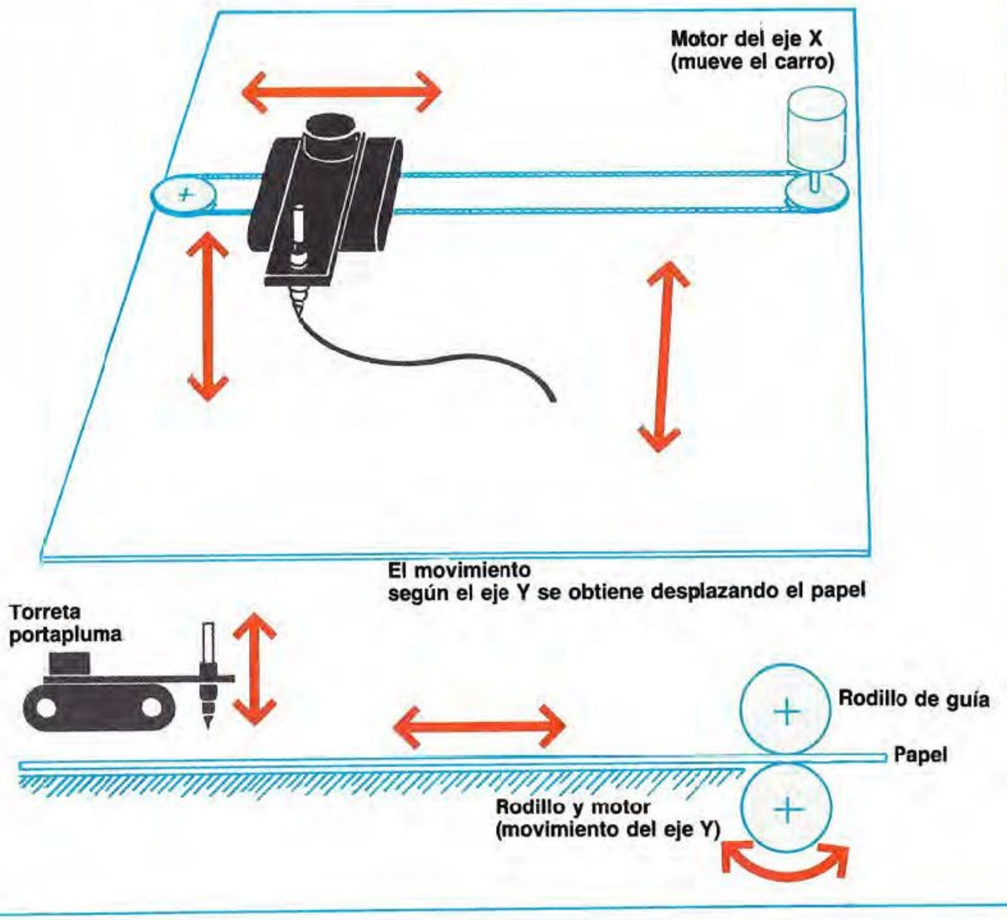
punto de coordenadas X,Y. En el segundo (pluma levantada) se tendrá un sencillo posicionado de la pluma, sin trazar ningún signo. Utilizando estos dos comandos es posible dibujar cualquier gráfico.

En algunos tipos de plotter plano falta el movimiento de la guía de uno de los dos ejes; el desplazamiento correspondiente se obtiene moviendo el papel (ver la figura de abajo).

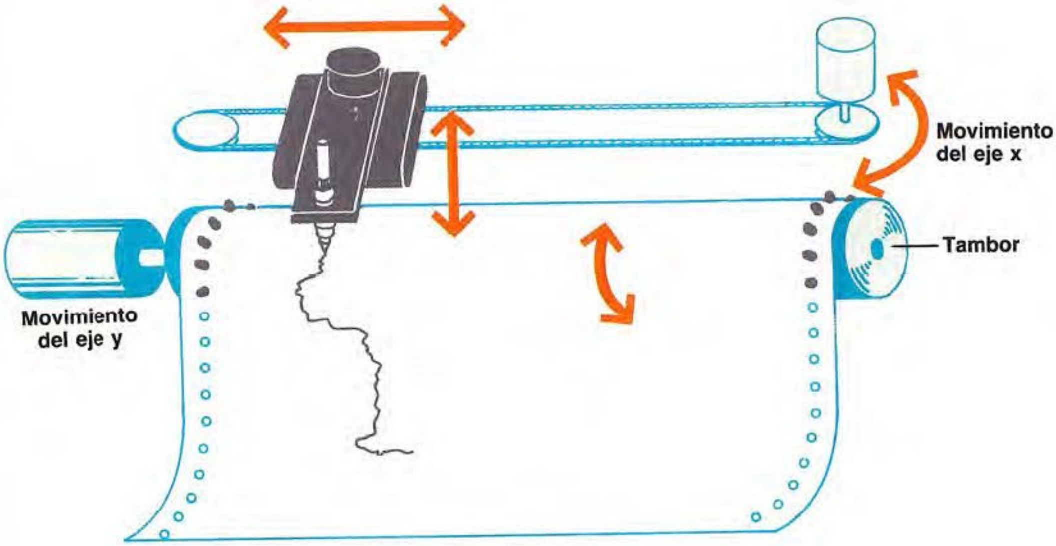
En el **plotter de tambor**, el movimiento según el eje Y se obtiene desplazando el papel como en la variante de los plotters planos; la diferencia con respecto a aquéllos reside en el método utilizado para obtener el desplazamiento del papel. Éste debe tener un taladrado de arrastre (como el papel de las impresoras) y su movimiento se obtiene con un rodillo dentado. Naturalmente, el rodillo debe poder girar en las dos direcciones horaria y antihoraria, de manera que pueda realizar los desplazamientos según los dos sentidos del eje Y. La principal ventaja de este tipo de plotter reside en la longitud no limitada del papel; sin embargo, se trata de máquinas mucho más caras que las anteriores y raramente empleadas con los pequeños sistemas de proceso.

Todos los plotters pueden disponer de más plumas con puntas de diferentes colores. Los métodos usados para seleccionar una de las plumas son dos. En el primer caso, todas las plumas están montadas al mismo tiempo sobre la torreta, y se baja (o sea dibuja) sólo la seleccionada. En el segundo caso, las diversas plumas están depositadas en adecuados alojamientos en los bordes del papel y el carro toma la seleccionada mediante un dispositivo adecuado. En ambos casos, el software de gestión del plotter, debe prever la instrucción de selección de la pluma. Debe prestarse atención al hecho de que el término «pluma» también se emplea para

VARIANTE DEL PLOTTER PLANO



PLOTTER DE TAMBOR



identificar la «pluma lógica», a la que no corresponde una pluma física (por ejemplo de diferente color), sino un diferente tipo de trazo, realizado siempre con la misma pluma física. Por tanto, se trata no de una pluma real, sino de una subrutina que controla el trazado de líneas con trazos diferentes. Otros dos factores muy importantes en la evaluación de las prestaciones de un plotter son la **resolución** y la **repetibilidad**. La resolución indica el mínimo desplazamiento que es posible presentar, determinado por la calidad de la mecánica y, en particular, de los motores. Estos últimos son de un tipo muy particular (motores de paso a paso), que pueden realizar una rotación muy pequeña (step o paso) para cada impulso eléctrico recibido; cuanto más pequeña es la rotación (paso) más pequeño es el correspondiente desplazamiento de la pluma. Como orden de magnitud, un plotter de calidad media permite desplazamientos mínimos de algunas décimas de milímetro. El segundo parámetro, la repetibilidad, indica con qué precisión es posible posicionar varias veces en el mismo punto. En otras palabras, con un par de valores numéricos enviados por el ordenador, que representan las coordenadas del punto, no corresponde siempre exactamente la misma posición física de la pluma. También en este caso, el error está generado por la mecánica del aparato y, normalmente, es inferior como valor a la resolución. En realidad, estos errores son menos determinantes de lo que podría parecer. En las aplicaciones generales, el resultado es más preciso que el obtenido a mano; para las aplicaciones especiales, como por ejemplo la preparación de circuitos impresos, el dibujo se traza a una escala mucho mayor, y una reducción fotográfica elimina los defectos. En estos casos, el uso del ordenador es realmente determinante, puesto que permite probar con varios factores de escala sin necesitar otras intervenciones. Para estas aplicaciones particulares existen plotters construidos adecuadamente, llamados **plotters fotográficos**, que utilizan un rayo de luz en lugar de la pluma; naturalmente, necesitan el empleo de material fotosensible (como el utilizado para la preparación de los circuitos impresos) y tienen una estructura mecánica particularmente estudiada para obtener precisiones muy elevadas.

El monitor monocromático. La tecnología empleada en la construcción de los monitores, y por tanto su estructura física, depende del tipo de aplicación a la que van destinados. La variedad de tipos existentes es notable, así como las diferencias de costo y de prestaciones.

El elemento fundamental de un vídeo es el tubo de rayos catódicos (TRC), un convertidor que puede transformar una señal eléctrica en una señal luminosa visible.

El funcionamiento del TRC se basa en las propiedades que tienen algunas sustancias (como las sales de fósforo) de emitir luz visible cuando en ellas incide un haz de electrones.

En la figura de la página siguiente se han esquematizado los dos tipos de TRC:

- TRC de deflexión electromagnética
- TRC de defexión electrostática

que difieren en el método adoptado para la deflexión del haz electrónico.

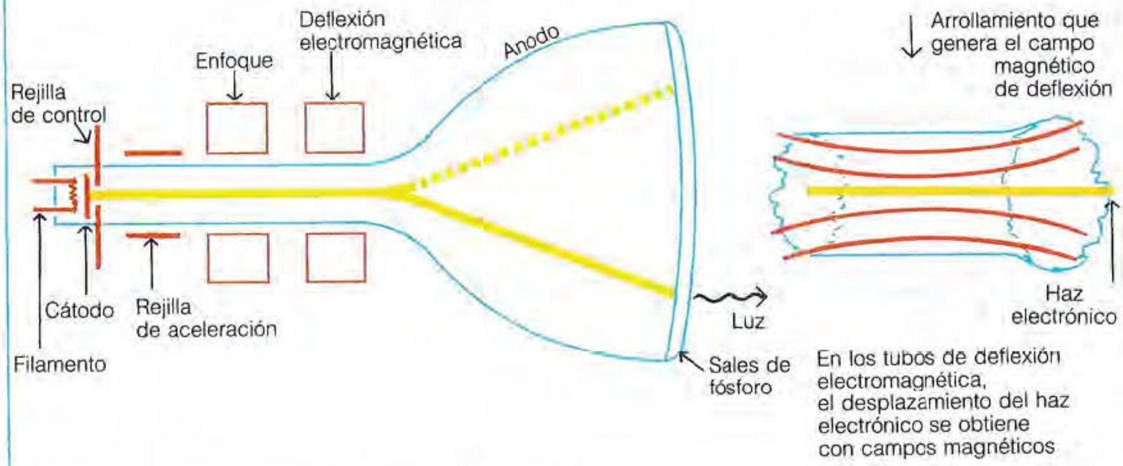
En ambos casos, por un filamento circula una corriente eléctrica que lo lleva a la incandescencia. En sus cercanías (normalmente es un pequeño tubo coaxial con el filamento) hay una fina capa de material particular que libera electrones al calentarse. Bajo el efecto del calentamiento generado por el filamento, este componente, llamado cátodo, emite electrones que son puestos en movimiento hacia el otro extremo del tubo (llamado ánodo) por una rejilla de aceleración. La cantidad de electrones enviados hacia el ánodo se regula mediante la rejilla de control, que con su potencial eléctrico hace las veces de elemento de regulación del flujo (como un grifo). Si no existiesen otros componentes, la nube de electrones producida por el cátodo iría a parar al ánodo y excitaría de manera desordenada las sales de fósforo depositadas sobre aquél; este material, golpeado por los electrones, produciría una luminosidad difusa en la pantalla. Por tanto, lo primero que debe hacerse es enfocar los electrones, y en el interior del TRC hay un electrodo de enfoque que tiene la misión de concentrar el haz de manera que llegue a golpear las sales de fósforo con una sección mínima (ver la figura de la pág. siguiente). Para completar el conjunto debe insertarse un sistema que permita desplazar el haz enfocado de electrones. Esto puede obtenerse con un campo magnético (deflexión magnética)

Impresora-plotter plana para colores.

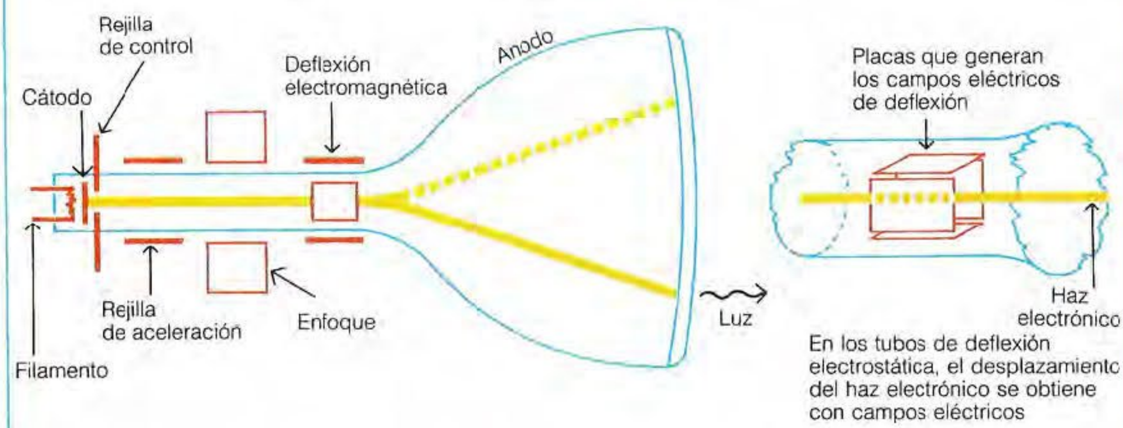


ESQUEMA DE UN TUBO DE RAYOS CATODICOS

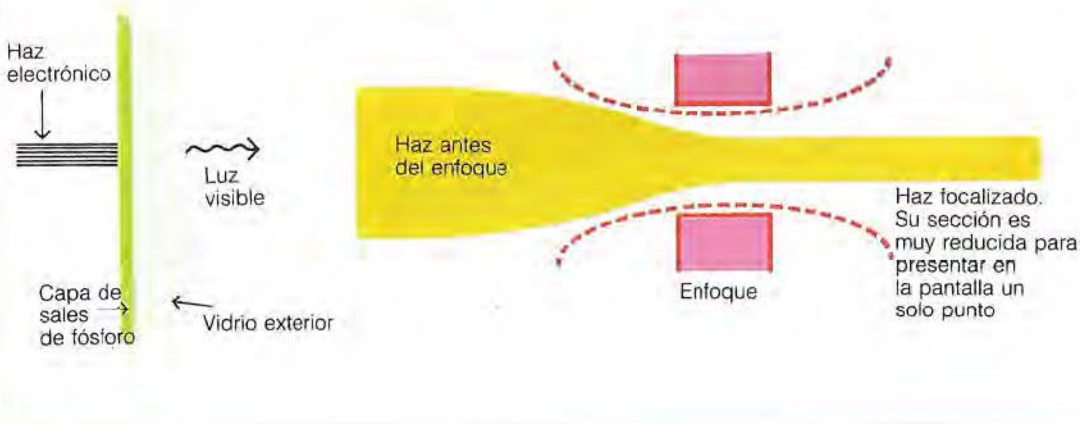
Deflexión electromagnética



Deflexión electrostática



Sistema de enfoque



o con un campo eléctrico (deflexión electrostática); en ambos casos se utilizan dos campos perpendiculares entre sí, de manera que puedan producirse desplazamientos del haz según los dos ejes (X,Y) para posicionarlo sobre cualquier punto de la superficie de la pantalla.
Los componentes eléctricos que regulan la generación, la aceleración y el enfoque del haz de electrones son gestionados por sistemas que no interaccionan con el ordenador, mientras que los elementos de deflexión son controlados por el ordenador; a través de adecuados interfaces, los datos enviados por el ordenador son convertidos en señales eléctricas que regulan la posición del haz sobre la pantalla.
La generación de una imagen se obtiene moviendo el haz electrónico de acuerdo con las instrucciones enviadas por el calculador en forma de señales de posicionado. También en este caso, como en los plotters, el ordenador debe especificar si un cierto desplazamiento debe ser un trazo visible o no.
La emisión de luz en un punto de la pantalla permanece mientras el haz electrónico excita aquel punto. Si se deben visualizar dos puntos alineados horizontalmente, en el instante en que el haz excita el segundo punto, los fósforos del primero ya no lo están. Como el trazado de un segmento se realiza excitando en sucesión los puntos que lo componen, es necesario evitar que, en el acto de la visualización de los últimos, los primeros hayan perdido luminosidad.
Esto puede obtenerse aprovechando la persistencia de la imagen, o sea la aptitud de las sales de fósforo de emitir luz durante un cierto tiempo después del instante de la excitación. Sin embargo, la persistencia sólo puede aprovecharse dentro de ciertos límites, porque se correría el riesgo de tener inadmisibles tiempos de borrado de la imagen.
Por tanto, al aprovechamiento de la persistencia se asocia la técnica del «refresco» que consiste en volver a dibujar la imagen varias veces por segundo. Generalmente se calibran la persistencia y la frecuencia de refresco para obtener imágenes de calidad aceptable. También la persistencia de la imagen sobre la retina del ojo debe entrar en la evaluación, que conduce a elegir en la mayor parte de los casos una frecuencia de refresco igual a 30 Hz. Esto significa que una imagen visualizada se redibuja completamente 30 veces por segundo.
El mantenimiento de la imagen con la técnica de refresco requiere el uso de una zona de memoria dedicada a la pantalla. El sistema operativo toma los datos de esta memoria y activa en consecuencia los diversos puntos del vídeo, repitiendo la lectura con una frecuencia de unas 30 veces por segundo. La desventaja de este método reside en la necesidad de dedicar al vídeo una notable área de memoria, que así queda sustraída a los programas de aplicación. En cambio, ofrece notables capacidades de gestión, puesto que el usuario puede modificar la presentación gráfica interviniendo directamente sobre las posiciones de memoria.
En algunas aplicaciones, esta técnica no puede utilizarse, por ejemplo cuando la resolución requerida hace demasiado grande el área que debe dedicarse al vídeo. En estos casos se emplea un TRC particular, llamado TRC «de memoria», que conserva la imagen durante tiempos notablemente prolongados. La operación de refresco ya no es necesaria, así como tampoco el área de memoria dedicada.
Sin embargo, este tipo de TRC presenta la notable desventaja de no permitir el borrado o la modificación de partes de la pantalla.
Con el método anterior (refresco), para variar un gráfico basta con modificar el contenido de la memoria; todo lo presentado en el vídeo desaparece varias veces por segundo, y aparece otras tantas veces con las modificaciones.
En los tubos de memoria, la imagen es fija por la persistencia, y no puede actualizarse hasta después de un borrado completo.
Una última clasificación de los monitores puede hacerse en base al método adoptado para la formación de la imagen.
Los principales métodos son dos:

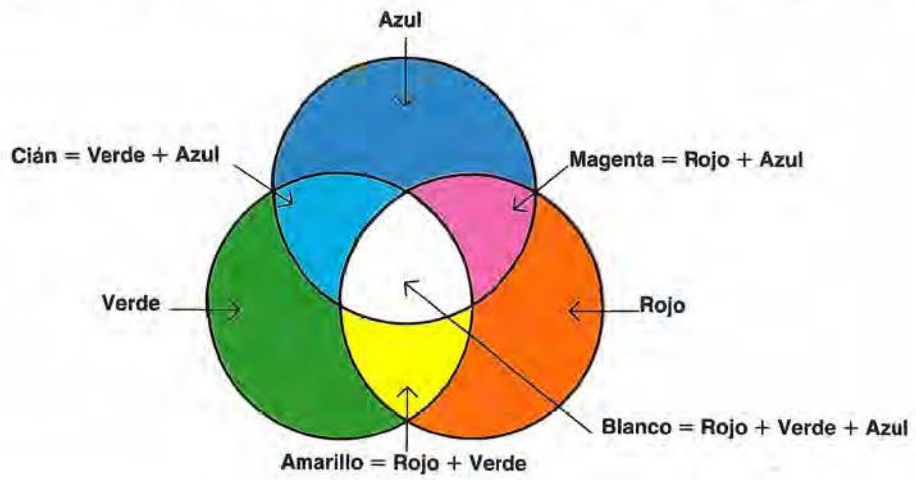
- por vectores, normalmente indicado con el término **stroke**
- por puntos, habitualmente llamado **raster**

También existen otros métodos que son variantes de los anteriores, pero son poco usados o se han abandonado. Por ejemplo, el método **starbust**, muy usado en el pasado, ya no se utiliza a causa de la escasa resolución, aunque todavía es válido en teoría para ahorrar memoria. La disminución de los costes de las memorias han hecho más adecuados los otros métodos.
El sistema stroke consiste en controlar el haz electrónico en una zona cualquiera de la pantalla, sin ningún orden de prioridad. Cada elemen-

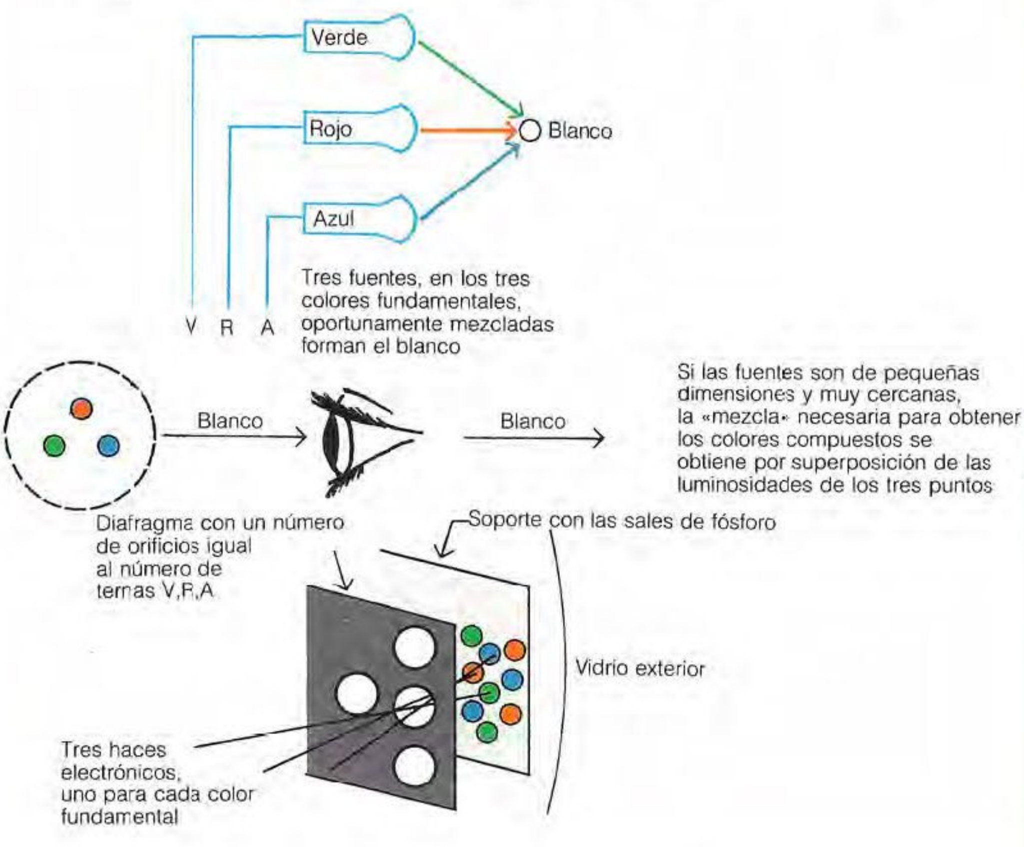
to de figura se obtiene moviendo el haz según una serie de desplazamientos elementales.
En el sistema raster, la pantalla es recorrida continuamente por el haz electrónico que, si estuviese siempre visible, trazaría una fina serie de líneas casi horizontales. Para obtener un punto en el vídeo se envía una señal, sincronizada con la posición del haz, que inhibe o activa la presentación. De esta manera, cada elemento de figura se reproduce activando selectivamente una matriz de puntos, de manera similar al sistema adoptado en las impresoras de agujas. La calidad gráfica depende directamente del número de puntos que puede direccionarse en la pantalla. Los valores típicos son, para la matriz de los caracteres, 5 × 7 o 7 × 9 puntos, para toda la pantalla de algunas decenas de miles hasta varios millones de puntos.
Normalmente, la resolución gráfica de la pantalla se indica especificando el número de puntos en horizontal y en vertical; por ejemplo, una pantalla de 280 × 190 puede visualizar 280 puntos consecutivos en horizontal y 190 en vertical, para un total de 53.000 puntos en la pantalla. Este grado de resolución es poco adecuado a las aplicaciones gráficas; los valores más usuales son del orden de magnitud de los 1000 × 1000 puntos, y en algunas aplicaciones más avanzadas, este límite se supera ampliamente.
La limitación impuesta al número de puntos de la pantalla depende de la construcción del tubo, de la granulosidad de las sales de fósforo y, sobre todo, de la capacidad de memoria.
Para tener un orden de magnitud, considérese una pantalla de 1000 × 1000. Como el estado de cada punto (activo/no activo) puede memorizarse en un bit, son necesarios por lo menos 10^6 bits. Para una máquina de 16 bits, cada posición de memoria puede contener hasta 16 bits y, por tanto, el área de memoria dedicada a la pantalla es de $10^6/16 = 62.500$ bytes, valor prácticamente igual a toda el área de memoria direccionable de una CPU de 16 bits y utilizada en los programas de aplicación. Para una máquina de 8 bits, las localizaciones de memoria necesarias son en cambio de 125.000, o sea cerca del doble de la capacidad de direccionamiento. Estas dificultades se traducen en limitaciones en la programación.
Las máquinas del tipo micro y personal pueden dividirse, a efectos de los gráficos, en dos categorías: las que para memorizar la pantalla de vídeo utilizan áreas de memoria que pertenecen al área del usuario y las que disponen de recursos independientes. Las primeras tienen la ventaja de la fácil accesibilidad a los datos gráficos por parte del usuario; la memoria está comprendida en el área direccionable del Intérprete o del Compilador y, por tanto, puede ser completamente gestionada por el programa. En cambio, el área de memoria disponible disminuye con la cantidad reservada a los gráficos. Por este motivo, normalmente no se han previsto valores elevados de resolución. En las segundas, la imagen se conserva en un área de memoria que no pertenece al área de usuarios, que de este modo conserva toda el área direccionable. Por el contrario, los otros gráficos sólo pueden escribirse y leerse utilizando subrutinas de sistema particulares, a través de instrucciones de alto nivel previstas en la máquina.

El monitor en colores. El principio de funcionamiento y las técnicas de formación y mantenimiento de la imagen son las mismas utilizadas en los tipos monocromáticos, mientras que varía notablemente la estructura física necesaria para su funcionamiento.
La formación de una imagen en colores puede realizarse de dos maneras. La técnica generalmente utilizada en las aplicaciones gráficas es el método aditivo (el mismo utilizado en los televisores en color), mientras que la técnica sustractiva se utiliza principalmente en fotografía o en los vídeos gráficos para los grandes sistemas. Los colores se dividen en dos categorías: los primarios y los compuestos. Se llaman primarios los colores que, combinados de diferente manera, permiten obtener los demás.
En la figura de arriba de la página siguiente se muestra la técnica de formación de algunos colores con el método aditivo, utilizando como colores primarios el rojo, el verde y el azul. En la figura se muestran sólo algunos de los colores secundarios (blanco, amarillo, cián y magenta), obtenidos por la adición de los colores fundamentales; los otros se generan variando las proporciones de adición: por ejemplo, el amarillo se obtiene mezclando verde y rojo en partes iguales, mientras que el anaranjado se obtiene aumentando el porcentaje de rojo.
En la figura de abajo de la página siguiente se ha esquematizado el principio utilizado para la formación de imágenes vídeo en colores. Un punto aparece blanco (o de cualquier otro color compuesto) si en él convergen los haces lumi-

TABLA DE COMPOSICION DE LOS COLORES CON TECNICA ADITIVA



FORMACION DE LOS COLORES EN UN VIDEO DE TRES HACES



nosos de tres pantallas diferentes, cada uno con emisión de luz en uno de los tres colores fundamentales. Análogamente puede tenerse una mezcla excitando partículas de los fósforos de los tres colores fundamentales y de dimensiones muy reducidas y cercanas entre sí, de manera que los tres haces de luz producidos sean perceptibles como un único haz.

Este segundo método es el más empleado en los TRC en colores. Tres dispositivos generan y controlan cada uno un haz electrónico que activa uno de los tres tipos de fósforo; regulando oportunamente el flujo de electrones sobre cada uno de los fósforos se tiene la generación de los colores compuestos. Un segundo tipo contiene un solo dispositivo de generación del haz electrónico (este detalle en los TRC suele indicarse con el término «cañón electrónico») mientras que la pantalla conserva la estructura de tres fósforos; la selección de los colores se consigue regulando la penetración del haz de electrones en los fósforos.

En el proyecto de los mecanismos de visualización de las imágenes en colores a menudo se presta atención a algunos efectos fisiológicos que permiten obtener buenos resultados ahorrando colores, y por tanto ocupar un área de memoria de dimensiones más reducidas para contener las informaciones necesarias a la gestión de la pantalla.

Las disposiciones principales (las mismas utilizadas en la transmisión televisiva en colores) tienen las siguientes características:

- Las áreas de dimensiones más grandes, correspondientes a las informaciones cromáticas principales, deben emplear los tres colores fundamentales.
- Las áreas más pequeñas pueden utilizar sólo dos colores.
- Las zonas de dimensiones mínimas pueden no estar coloreadas, puesto que por debajo de una cierta dimensión, el ojo no consigue percibir los colores, sino sólo el brillo.

Una sugestiva imagen de las tarjetas de un monitor en colores.



Ericsson

La cantidad de memoria empleada en un vídeo en colores es notablemente mayor que la necesaria para el vídeo monocromático. En este último, un solo bit indica el estado de un punto de la pantalla, puesto que el propio estado físico del punto sólo puede asumir dos valores (iluminado o apagado).

En el caso de las pantallas en colores, para cada punto debe memorizarse también la composición cromática, o sea cuáles son los colores fundamentales presentes en el punto considerado. Como resultado, los colores que pueden obtenerse, en la práctica están limitados a la capacidad de memoria, y en los sistemas de calidad media se adoptan cuatro u ocho colores.

Para memorizar cuatro posibles combinaciones de los colores fundamentales son necesarios dos bits por cada punto del vídeo (con dos bits pueden escribirse los valores binarios 00, 01, 10, 11, correspondientes a los decimales 0, 1, 2, 3), con lo que se duplica la memoria necesaria con respecto al caso monocromático.

En los sistemas de ocho colores, los bits necesarios son tres, y se triplica la memoria ocupada (en una pantalla de 1000 × 1000 se ocupan cerca de 180 kbytes de 16 bits).

Para memorizar una cantidad tan grande de informaciones deben emplearse varias páginas de memoria, cada una de una amplitud igual al número de puntos de la pantalla, y el número de páginas debe ser igual al número de bits reservados para las combinaciones cromáticas. Así, para la representación en cuatro colores son necesarias dos páginas, para las de ocho colores, tres páginas. Normalmente, el sistema utiliza para el color también el área de mapa monocromático de modo que, respecto al caso monocromático, para cuatro colores es necesaria una sola expansión de memoria, y para ocho dos expansiones.

El sistema operativo considera estas páginas de memoria como matrices cuya composición genera, para cada punto de la pantalla, la mezcla de los colores fundamentales requeridos.

En otros tipos de máquinas, el área de memoria destinada a los gráficos ya viene dimensionada para acoger los indicadores de color y no está dividida en páginas. Esta estructura no se hace evidente para el usuario si éste emplea las instrucciones de alto nivel previstas en el lenguaje de la máquina, mientras que puede crear complicaciones en el direccionamiento de la memoria trabajando en Assembler, o bien en DMA.

Otro problema surge de la transposición del gráfico sobre papel. Normalmente, los plotters o las impresoras gráficas no disponen de más de dos o tres colores. Un método para obtener resultaldos aceptables consiste en reproducir las diversas tonalidades con impresiones sucesivas, cada una en uno de los colores disponibles, con un símbolo que ocupa más o menos espacio en función del porcentaje en que debe obtenerse aquel color. Por ejemplo, utilizando para un cierto color el símbolo ., su porcentaje en el dibujo será bajo, mientras que empleando un símbolo que llene en gran parte la matriz, por ejemplo X, se tiene un aumento del porcentaje. Intercalando muy repetidamente los diversos símbolos en los colores disponibles puede tenerse la sensación de un dibujo con más colores de los que realmente se han empleado.

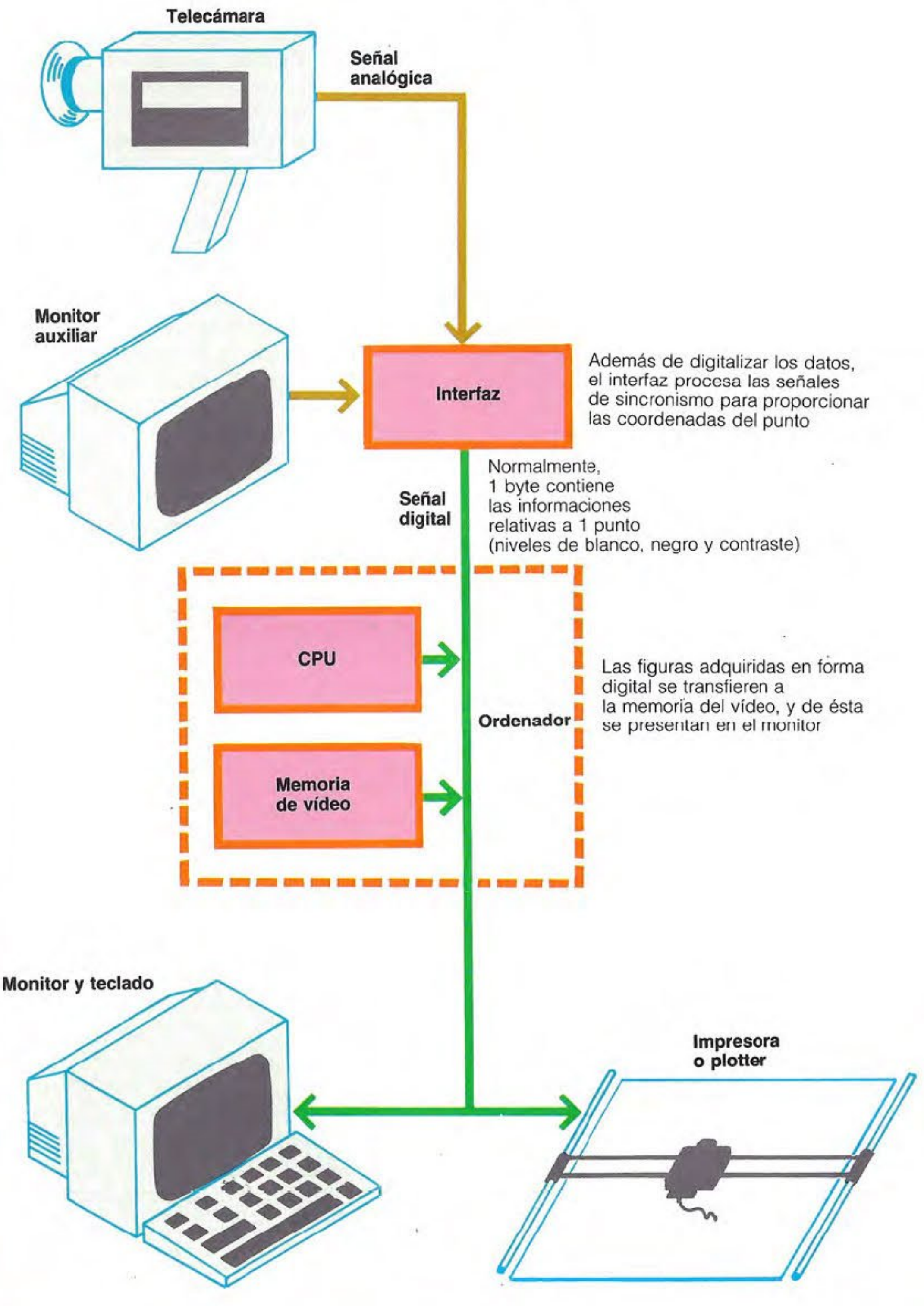
Hardware especializado

En los gráficos de ordenador existen componentes de hardware o sistemas completos dedicados a un determinado sector de aplicación o a un simple problema específico. Por ejemplo, para la adquisición de imágenes de una cierta complejidad (fotografías, dibujos) existen adecuados interfaces para la conexión del ordenador y una telecámara. De esta manera, la digitalización de la imagen es automática y permite la adquisición de figuras muy complejas en un tiempo muy breve. Los campos de aplicación de esta técnica son de lo más diverso, por ejemplo: los gráficos publicitarios, el análisis de datos meteorológicos transmitidos por telefoto vía satélite, la radiología, etc.

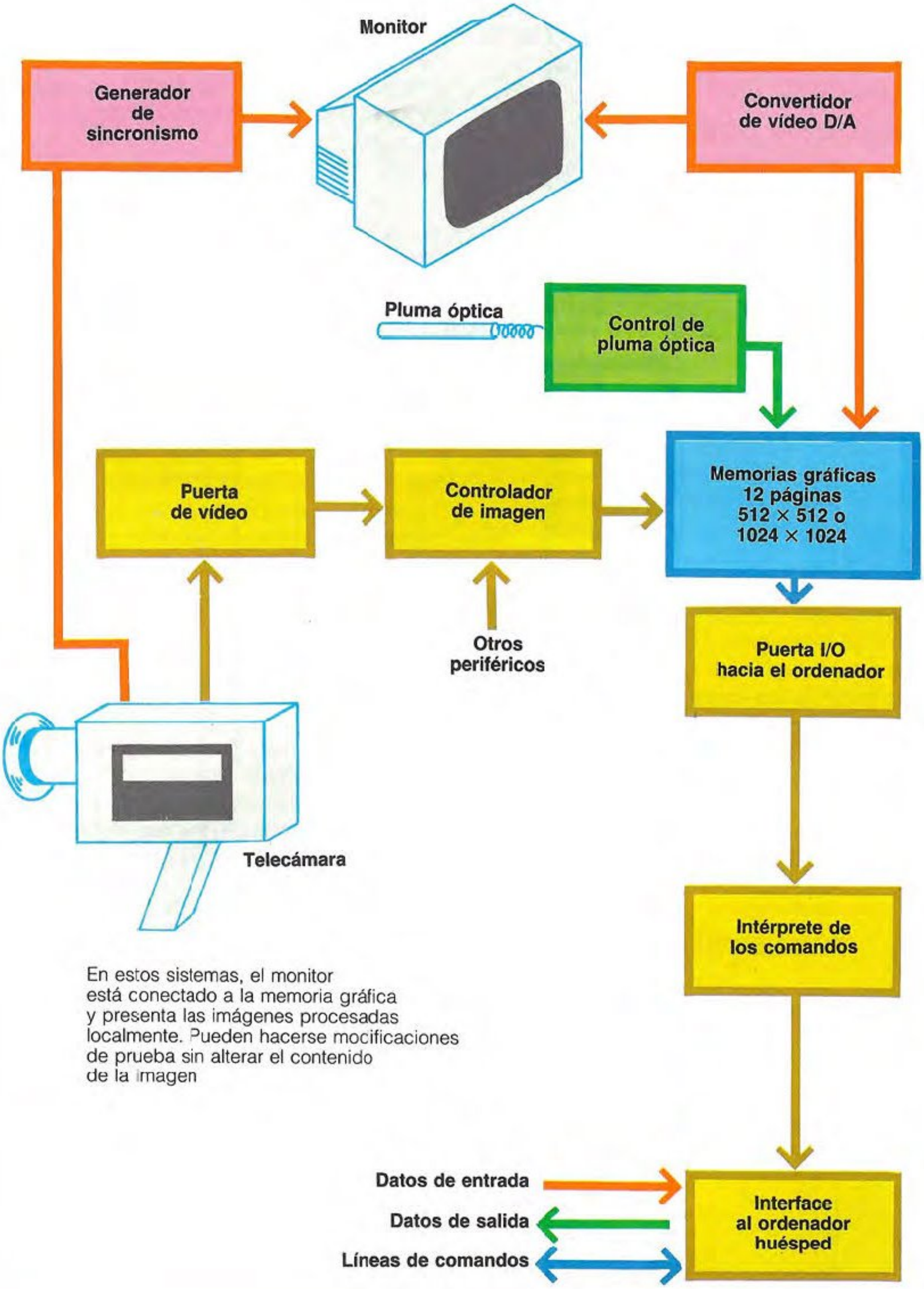
El esquema de la figura de la página siguiente se refiere a una aplicación sobre ordenador personal, mientras que para las máquinas de mayor potencia hay prevista una metodología diferente. En las pequeñas máquinas, el tratamiento de la imagen lo realiza enteramente la CPU, mientras que en las aplicaciones más complejas se utiliza un **controlador gráfico** que realiza él solo una gran parte de los procesos, utilizando el ordenador sólo como supervisor. El esquema de principio de un **controlador de pantalla gráfica** se muestra en la figura de la página 1413 y, a continuación, se relacionan algunos órdenes de magnitud correspondientes a las capacidades de estos sistemas

- resolución: de 512 × 512 a 1024 × 1024 (aproximadamente 1.000.000 de puntos)

ADQUISICION DE IMAGENES COMPLEJAS



CONTROLADOR DE PANTALLA GRAFICA



En estos sistemas, el monitor está conectado a la memoria gráfica y presenta las imágenes procesadas localmente. Pueden hacerse modificaciones de prueba sin alterar el contenido de la imagen

- páginas gráficas: hasta 12 (los valores típicos son 6/8)
- colores: de 64.000 a 16.000.000 (puede obtenerse un número de difuminados prácticamente infinito)
- velocidad de transferencia de datos: cerca de 1.000.000 de pixels por segundo

Instrucciones Basic para los gráficos

La forma y la variedad de las instrucciones dedicadas a los gráficos dependen estrechamente de las versiones del Basic utilizado. Como en la versión estándar no hay previstas instrucciones gráficas, no existe ninguna forma de referencia, y cada máquina utiliza simbologías propias, generalmente diferentes entre sí. Esto se debe principalmente a la estrecha interdependencia que existe entre el funcionamiento de este tipo de instrucciones y la estructura hardware de la máquina. La gestión de un vídeo en modo gráfico es notablemente diferente de la gestión de los caracteres y comporta algunas dificultades en los direccionamientos de la memoria, junto a las complejidades que se derivan de la eventual presencia de los colores. Sin embargo, pueden definirse metodologías generales aplicables en la mayor parte de los casos.

En una aplicación gráfica cualquiera es necesario disponer de una serie de instrucciones o de subrutinas que ejecuten algunas funciones elementales; el gráfico, cualquiera que sea, podrá obtenerse combinando oportunamente estas funciones. La función más importante es el trazado de un segmento dados los puntos extremos, ya que con ella pueden generarse todas las demás figuras.

Los ejemplos que se presentan a continuación se han realizado sobre dos tipos de máquina: el módelo 2010 de Siprel y el personal Olivetti M20. La primera utiliza instrucciones comunes al Apple y a muchas máquinas compatibles con Apple, mientras que la segunda tiene instrucciones específicas.

Trazado de segmentos

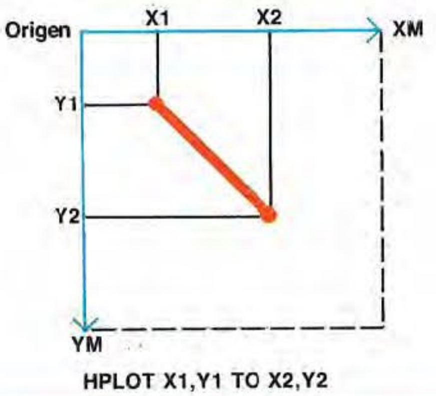
La pantalla vídeo (o el televisor) constituye el periférico más empleado en los gráficos, y precisamente por este motivo, la mayor parte de las instrucciones se refieren al vídeo. Para los demás periféricos (impresora gráfica, plotter, etc.), salvo casos particulares no pueden encontrarse fácilmente rutinas de gestión, y el usuario debe proceder a su escritura.

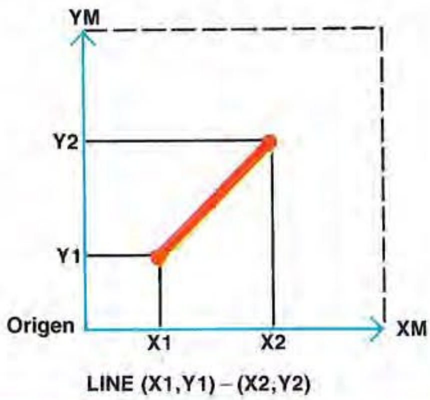
En las aplicaciones gráficas, cada punto de la pantalla de vídeo está identificado por un par de coordenadas. Así, para dibujar por ejemplo un segmento, es necesario especificar las coordenadas de los dos extremos. Las instrucciones de base son del tipo:

HPLOT X1,Y1 TO X2,Y2	(Siprel 2010, Apple y compatibles)
LINE (X1,Y1)-(X2,Y2)	(Olivetti M20)

Las citadas instrucciones son en realidad más complejas, y prevén algunas opciones que se ilustrarán más adelante. En la figura de abajo se han esquematizado las dos elecciones más utilizadas para los sistemas de coordenadas del ví-

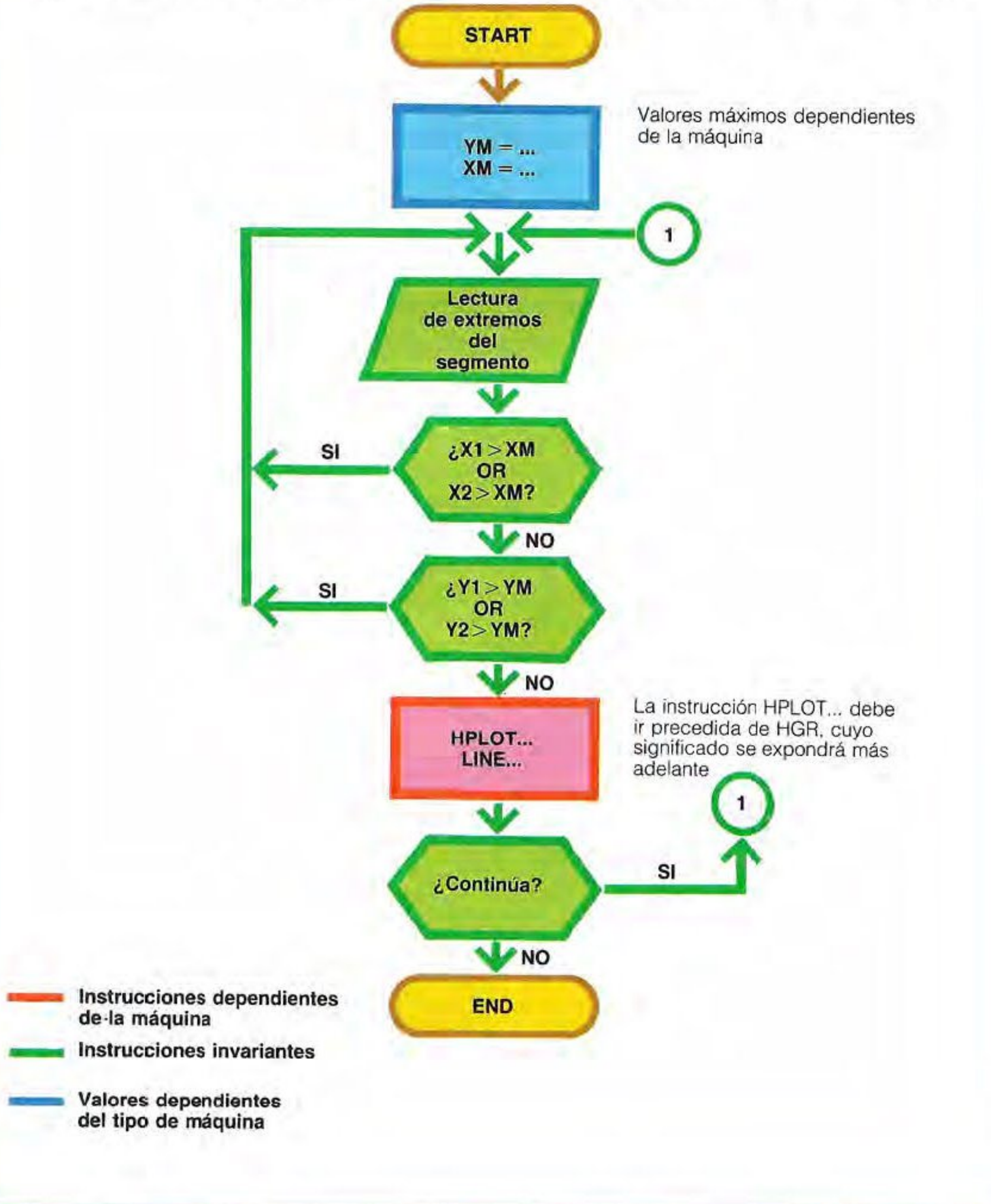
DIMENSIONADO DE LAS PANTALLAS DE VIDEO UTILIZADAS





deo; la diferencia reside en la orientación del eje y. Esto debe tenerse presente cuando en las instrucciones se citan las coordenadas de los puntos que interesan.
En la primera máquina (Siprel 2010), el origen de los ejes es arriba a la izquierda, en la segunda (M20) es abajo a la izquierda; el eje x siempre tiene el mismo sentido. En la primera máquina, aumentando el valor de y, el extremo inicial del segmento se desplaza hacia abajo y, en la segunda, hacia arriba. En la figura de abajo se muestra el diagrama de flujo de un programa que puede dibujar un segmento que une dos puntos definidos mediante las coordenadas. La única particularidad de estos programas es el control de los valores introducidos, destinado a

DIAGRAMA DE FLUJO DEL PROGRAMA PARA DIBUJAR UN SEGMENTO



TRAZADO DE UN SEGMENTO

Versión Siprel 2010

```
10 HOME
20 XM=279: YM=191
30 PRINT "Coordenadas iniciales";
40 INPUT X1,Y1
50 PRINT "Coordenadas finales";
60 INPUT X2,Y2
70 IF X1>XM OR X2>XM OR Y1>YM OR Y2>YM THEN GOTO 10
80 REM ***** Dibujo *****
90 HGR: HCOLOR=3
100 HPLOT X1,Y1 TO X2,Y2
110 PRINT "Continuo (s/n)";
120 INPUT A$
130 IF A$="s" THEN GOTO 10
140 END
```

Versión Olivetti M20

```
10 CLS
20 XM=511: YM=255
30 PRINT "Coordenadas iniciales";
40 INPUT X1,Y1
50 PRINT "Cooroderadas finales";
60 INPUT X2,Y2
70 IF X1>XM OR X2>XM OR Y1>YM OR Y2>YM THEN GOTO 10
80 REM ***** Dibujo *****
90 REM No es necesaria
100 LINE (X1,Y1) - (X2,Y2)
110 PRINT "Continuo (s/n)";
120 INPUT A$
130 IF A$="s" THEN GOTO 10
140 END
```

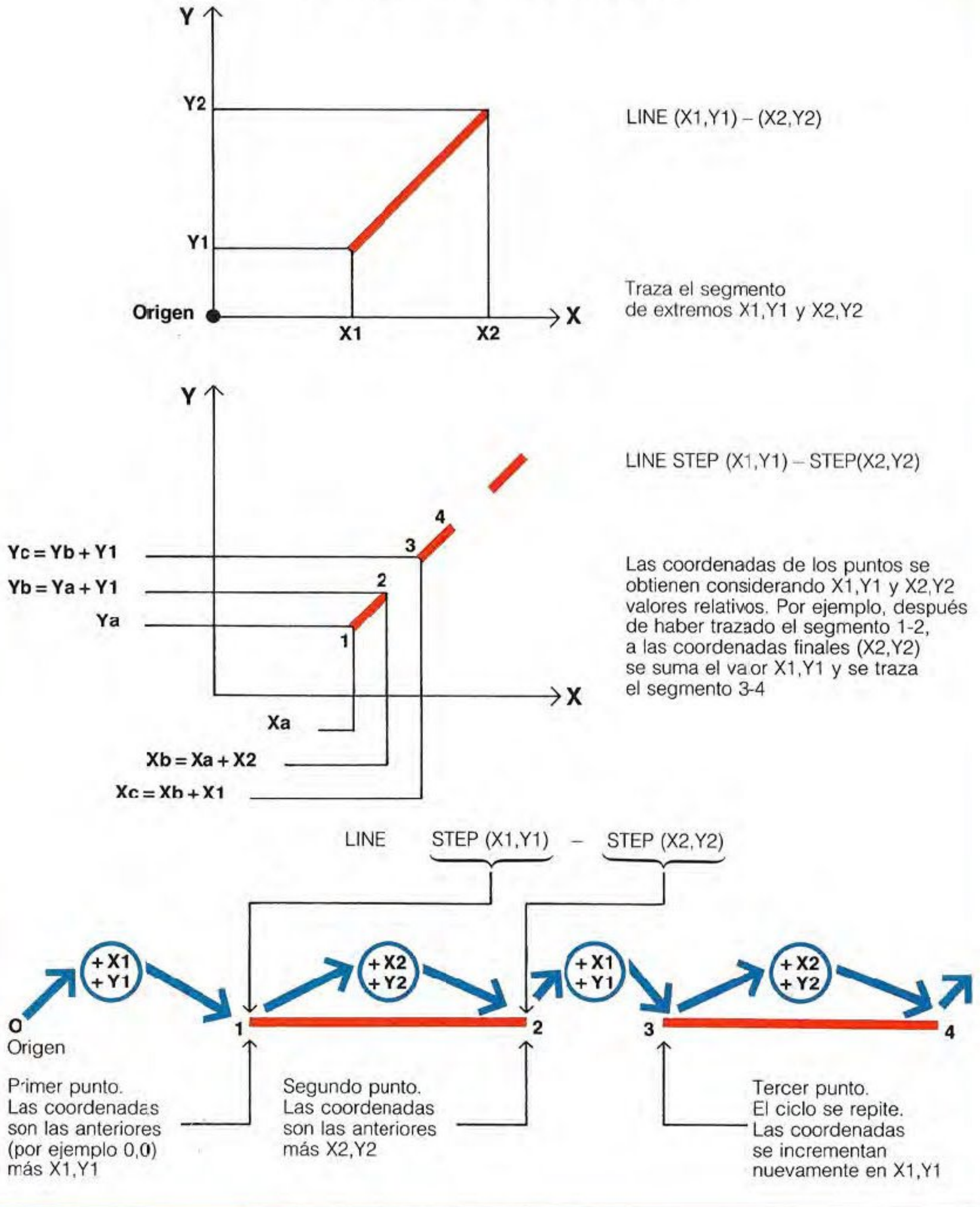
comprobar que las coordenadas de los extremos no superen el campo de la pantalla.
Arriba se han representado los listados de los dos programas para las dos máquinas indicadas. La diferencia principal es la instrucción que traza el segmento en el vídeo (línea 100). Además, en la primera máquina debe activarse la modalidad gráfica con la instrucción HGR (línea 90), cuyo significado se ilustrará más adelante.
La forma completa de la instrucción LINE de la Olivetti M20 es la siguiente

LINE %N STEP(X1,Y1)–STEP(X2,Y2),C,B,F, opción

donde

- LINE — es la instrucción general para trazar un segmento entre dos puntos de coordenadas (X1,Y1) y (X2,Y2)
- %N — es un valor que indica una de las ventanas en que es posible dividir la pantalla. Si se especifica este parámetro, la instrucción LINE sólo trabaja en la ventana correspondiente
- STEP — es la palabra clave para indicar el uso de coordenadas relativas. Utilizando la opción STEP, las coordenadas X1,Y1 se convierten en relativas al último posicionado efectuado. X2,Y2 son los nuevos valores iniciales. En la figura de la página siguiente se ha esquematizado el funcionamiento de la STEP
- C — indica el color con el que debe trazarse el segmento
- B — es un parámetro opcional con el que es posible trazar en lugar del segmento de extremos X1,Y1 y X2,Y2 un rectángulo con una diagonal cuyos extremos tienen las mismas coordenadas
- F — parámetro que permite llenar con el color especificado en C la superficie del rectángulo trazado con B.

EFECTO DE LA OPCION STEP



Naturalmente, la opción F sólo puede estar si existe la B

opción puede asumir uno de los valores AND, XOR, OR, NOT, PSET, PRESET, y sólo se utiliza con el color. Su significado y algunos ejemplos se ilustrarán más adelante.

La opción B libera al programador de la necesidad de escribir la rutina para el trazado de rectángulos como conjuntos de segmentos. En otras máquinas puede obtenerse el mismo resultado aprovechando la posibilidad de direccionado múltiple de la instrucción HPLOT.

La forma básica

HPLOT X1,Y1, TO X2,Y2

puede ampliarse para obtener sucesivos posicionados utilizando la opción TO repetida para cada nuevo par de coordenadas.
Por ejemplo, la forma

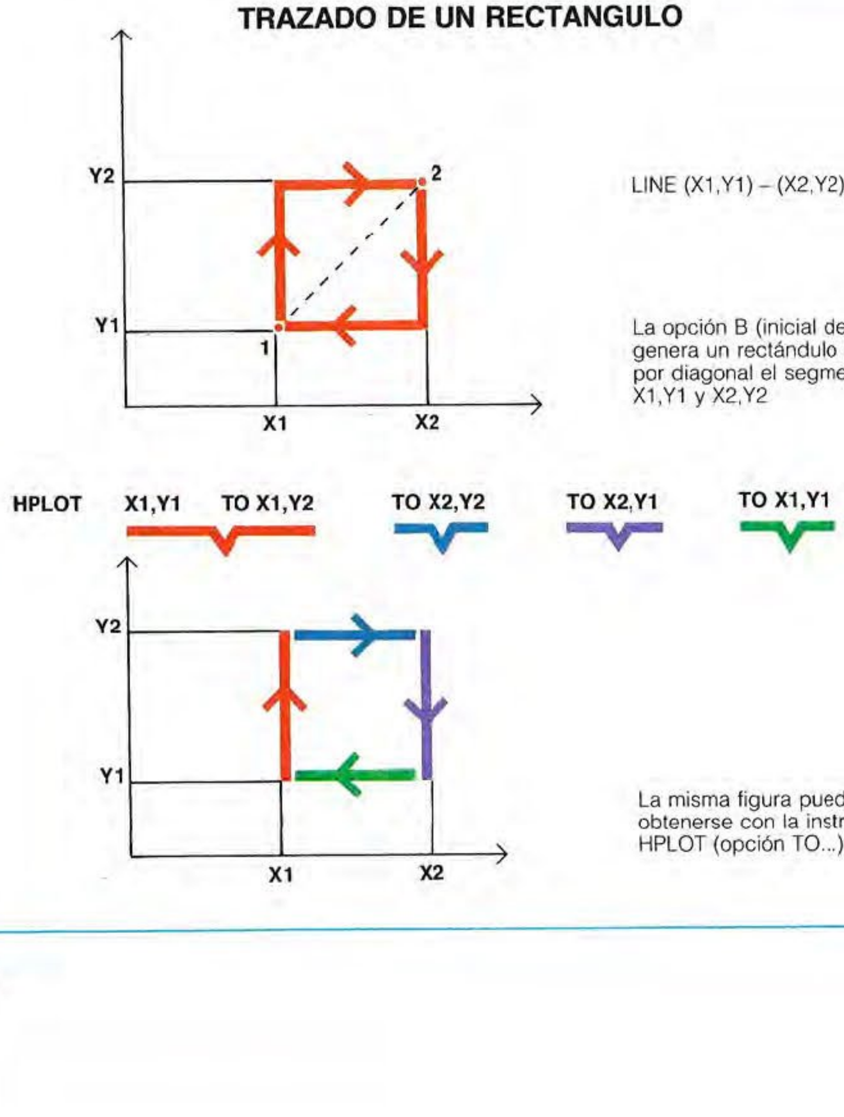
HPLOT X1,Y1 TO X2,Y2 TO X3,Y3 TO X4,Y4

puede utilizarse para trazar un rectángulo que tiene por vértices los puntos de coordenadas X1,Y1, etc., y por tanto se tiene un efecto similar al de la instrucción anterior con la opción B. En la figura de abajo se comparan los dos tipos de instrucción. Obsérvese que utilizando la LINE con la opción B, el rectángulo que se obtiene tiene siempre los lados paralelos a los bordes del vídeo (o de la ventana), mientras que la HPLOT no presenta esta limitación y puede utilizarse para trazar una figura general compuesta por un número cualquiera de segmentos. En la figura de la página siguiente se ha reproducido el diagrama de flujo de un programa que dibuja cuadrados y rectángulos en un punto cualquiera de la pantalla; además, se ha previsto la posibilidad de variar las dimensiones de la figura especificando un factor de escala. Las parametrizaciones se obtienen refiriendo todas las coordenadas de la figura a los valores X0,Y0 del vértice de posicionado y calculando los desplazamientos necesarios para obtener los lados. En las figuras de las páginas 1420 y 1421, esta lógica se aplica al trazado de un rectángulo. En las páginas 1421 y 1422 se han representado los listados del mismo programa en las dos versiones Siprel 2010 (Apple y compatibles) y Olivetti M20.

Rotación de un segmento

En muchas aplicaciones gráficas debe disponerse de subrutinas que permitan el traslado o la rotación de una figura. El traslado puede ob-

TRAZADO DE UN RECTANGULO

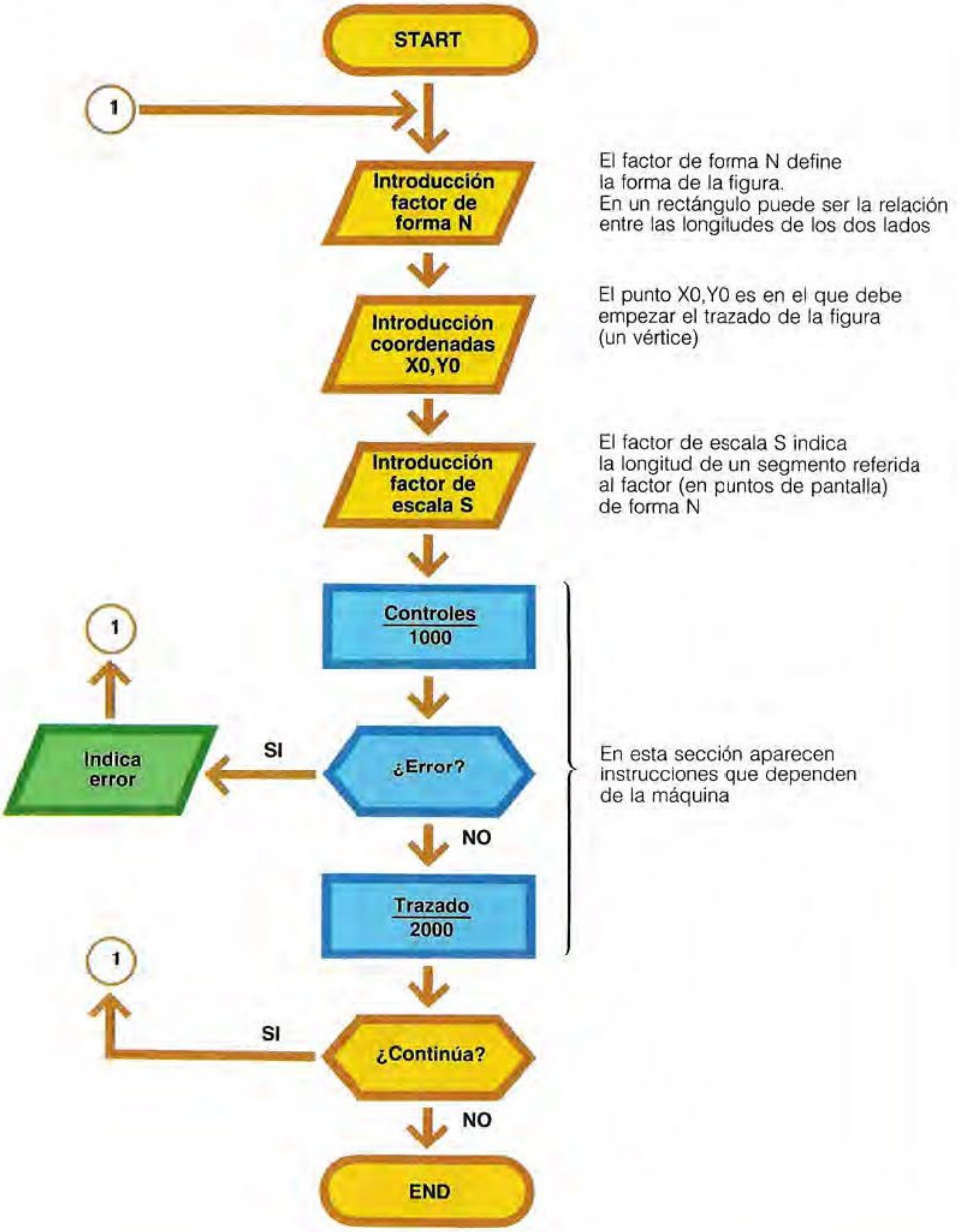


LINE (X1,Y1) – (X2,Y2) B

La opción B (inicial de la palabra box) genera un rectándulo que tiene por diagonal el segmento de vértices X1,Y1 y X2,Y2

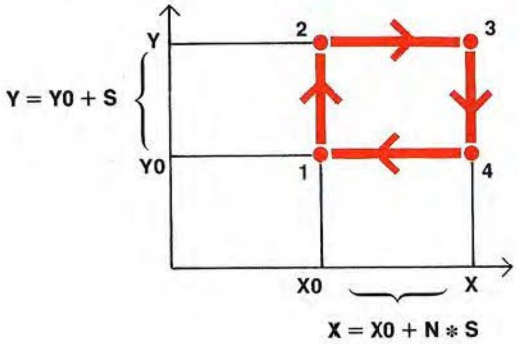
La misma figura puede obtenerse con la instrucción HPLOT (opción TO...)

DIAGRAMA DE FLUJO PARA EL TRAZADO PARAMETRIZADO DE CUADRADOS Y RECTANGULOS



tenerse muy sencillamente refiriendo todas las instrucciones que trazan los lados de la figura a un punto suyo (normalmente un vértice); al variar las coordenadas de este punto también variará la posición de todos los lados y así se obtendrá un nuevo posicionado de la figura. El método requiere parametrizar los desplazamientos necesarios para obtener cada uno de los lados, de manera idéntica al método utilizado en los diagramas de flujo anteriores para el cambio de

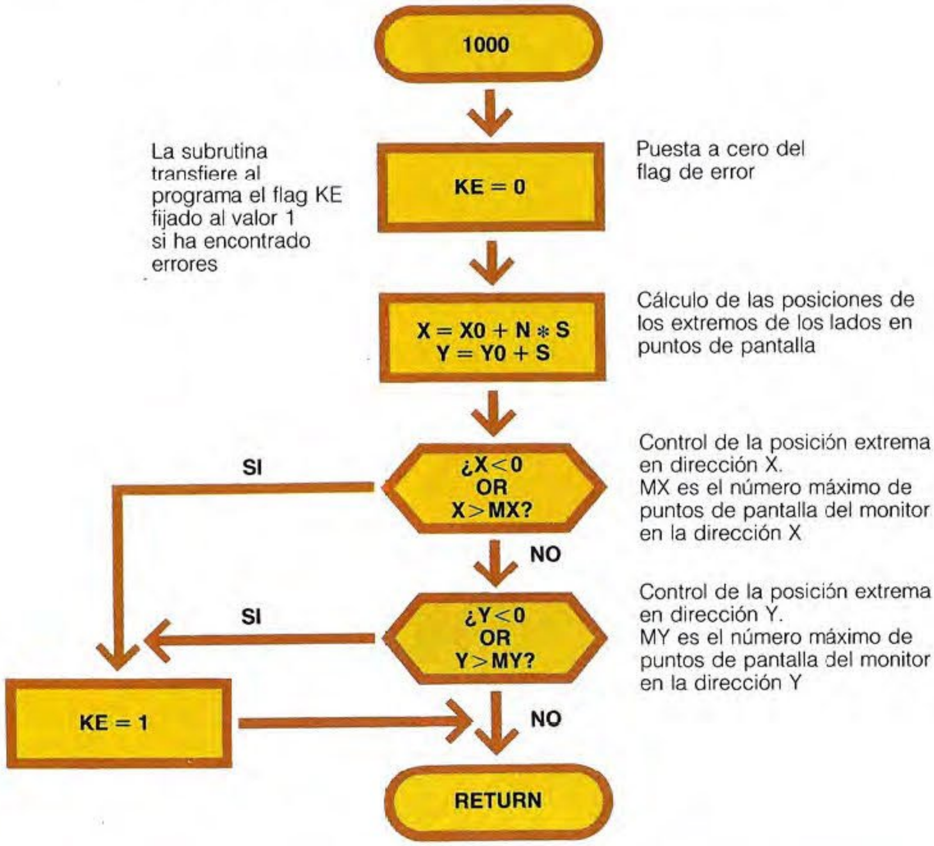
PARAMETRIZACION DE UNA FIGURA PLANA



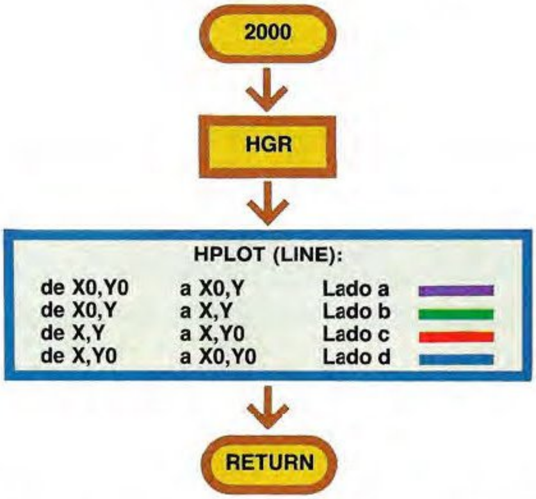
Los desplazamientos 1-2 y 3-4 dependen de la orientación del eje y. Si el origen es arriba a la izquierda se tiene Y = Y0 – DY.
En cambio, si el origen es abajo, se tiene Y = Y0 + DY

El parámetro N es el factor de forma y define la forma de la figura. Por ejemplo, puede ser la relación entre las dos dimensiones; en este caso, poniendo N = 1 se tendrá un cuadrado, mientras que con N = 2 se tendrá un rectángulo con la base de doble longitud que la altura. El parámetro S es el factor de escala, y proporciona la medida de cada lado en puntos de pantalla y referida al factor de forma.
Por ejemplo, poniendo N = 1 y S = 20, se tendrá un cuadrado con lados de longitud igual a 20 puntos de pantalla; poniendo N = 5 se tendrá un reactángulo con el lado menor (el paralelo al eje y) de 20 puntos de pantalla de longitud y el otro de 5 * 20 = 100 puntos de pantalla de longitud.

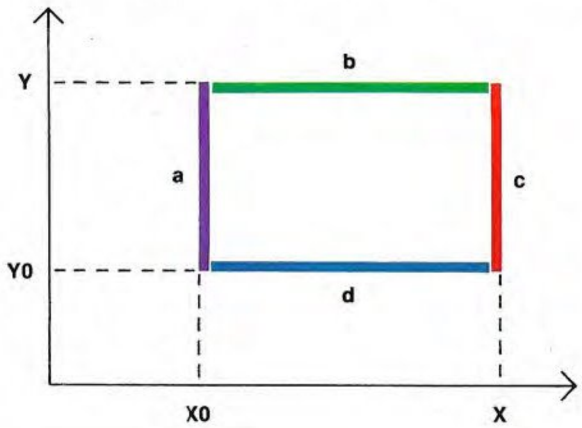
CONTROL SOBRE LOS DATOS INTRODUCIDOS



TRAZADO DE LA FIGURA



Las coordenadas X,Y se calculan en la subrutina 1000.
Como la figura tiene los lados paralelos a los ejes, no son necesarios otros puntos



TRAZADO PARAMETRIZADO DE CUADRADOS Y RECTANGULOS

Versión Siprel 2010, Apple y compatibles

```
5 MX=279: MY=191
10 TEXT: REM Selecciona la página de texto
15 HOME: REM Borra la pantalla
20 PRINT "Factor de forma";
30 INPUT N
40 PRINT "Coordenadas de referencia";
50 INPUT X0,Y0
60 PRINT "Factor de escala";
70 INPUT S
80 GOSUB 1000
90 IF KE=0 THEN GOTO 100
95 PRINT "Error": FOR P=1 TO 2000: NEXT P: GOTO 10
100 GOSUB 2000
105 VTAB 22: REM Posiciona el cursor en la línea n. 22
110 PRINT "Quiere continuar (s/n) ";
```

```
120 INPUT A$
130 IF A$="s" THEN GOTO 10
140 END
1000 REM ***** Controles *****
1010 KE=0
1020 X=XO+N*S: Y=YO+S
1030 IF Y>MY OR Y<0 OR X>MX OR X<0 THEN KE=1
1040 RETURN
2000 REM ***** Dibuja *****
2010 HGR: HCOLOR= 3
2020 HPLOT XO,YO TO XO,Y
2030 HPLOT XO,Y TO X,Y
2040 HPLOT X,Y TO X,YO
2050 HPLOT X,YO TO XO,YO
2060 REM Habría bastado la sola instrucción:
2070 REM HPLOT XO,YO TO XO,Y TO X,Y TO X,YO TO XO,YO
2080 RETURN
```

Versión Olivetti M20

```
5 MX=511: MY=255
10 REM No es necesaria
15 CLS: REM Borra la pantalla
20 PRINT "Factor de forma";
30 INPUT N
40 PRINT "Coordenadas de referencia";
50 INPUT XO,YO
60 PRINT "Factor de escala";
70 INPUT S
80 GOSUB 1000
90 IF KE=0 THEN GOTO 100
95 PRINT "Error": FOR P=1 TO 2000: NEXT P: GOTO 10
100 GOSUB 2000
105 REM No es necesaria
110 PRINT "Quiere continuar (s/n) ";
120 INPUT A$
130 IF A$="s" THEN GOTO 10
140 END
1000 REM ***** Controles *****
1010 KE=0
1020 X=XO+N*S: Y=YO+S
1030 IF Y>MY OR Y<0 OR X>MX OR X<0 THEN KE=1
1040 RETURN
2000 REM ***** Dibuja *****
2010 REM No es necesaria
2020 LINE (XO,YO) - (XO,Y)
2030 LINE (XO,Y) - (X,Y)
2040 LINE (X,Y) - (X,YO)
2050 LINE (X,YO) - (XO,YO)
2060 REM Habría bastado la sola instrucción:
2070 REM LINE (XO,YO) - (X,Y),1,B
2080 RETURN
```

escala. Por ejemplo, observando la figura de la página 1420, si se varían las coordenadas X0,Y0 dejando inalterados los parámetros N y S, la figura se trazará a partir (vértice 1) del punto definido por el nuevo par de coordenadas X0,Y0. Actuando de esta manera se obtiene el desplazamiento de la figura a una zona diferente de la pantalla, pero sin anular la eventual figura trazada anteriormente en otra posición. Para eliminarla debe borrarse toda la pantalla, o pasar por encima de la figura una «goma de borrar». La primera solución no siempre puede adoptarse, puesto que se borrarían también otras figuras eventualmente presentes que deben permanecer visibles. La segunda solución, en muchos casos es la única aceptable. En la modalidad gráfica pueden utilizarse generalmente varios colores, y entre éstos existe el color de fondo (background), indicado habitualmente con el código numérico 0. Este color pue-

de utilizarse como goma de borrar, puesto que repasando las líneas de una figura con el color del fondo, el resultado es la desaparición de la figura. El código 0 para indicar el color de fondo negro (en la pantalla, el dibujo está trazado en blanco sobre negro) es válido para casi todas las máquinas, pero en casos particulares pueden existir códigos numéricos diferentes. La instrucción que permite implantar un color normalmente es

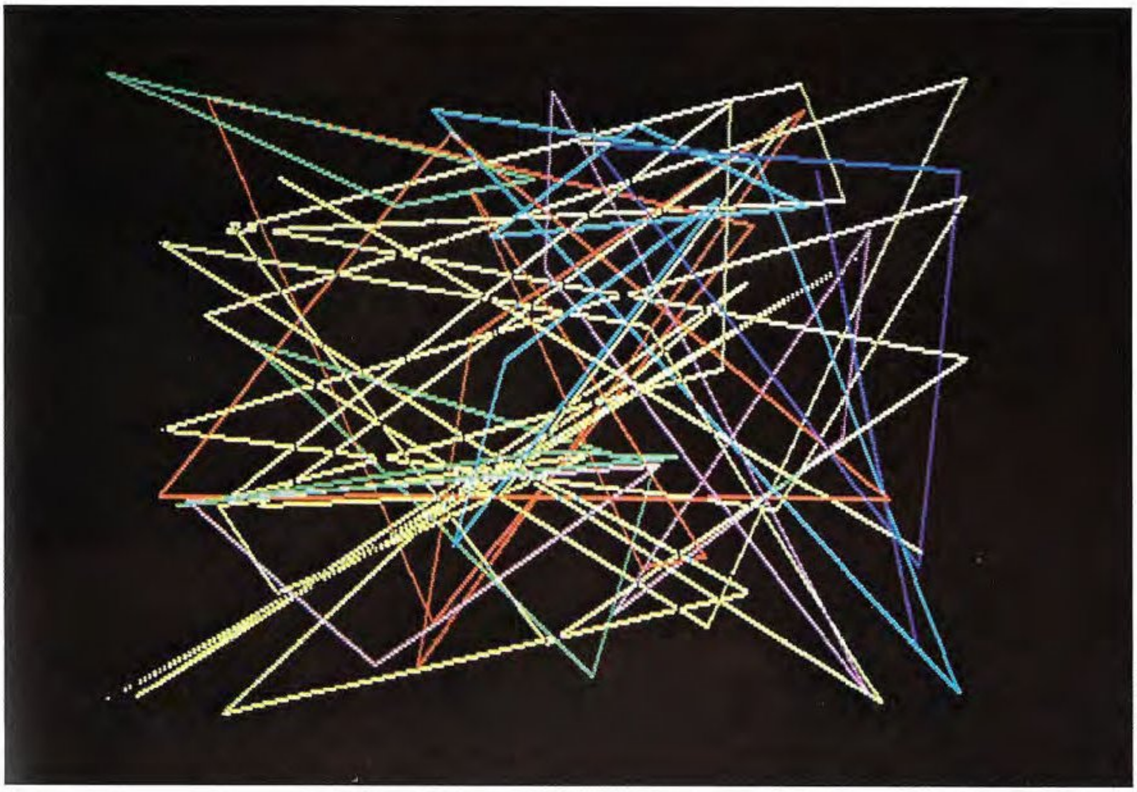
COLOR = n

donde n es el código del color deseado; en algunos casos debe omitirse el símbolo =. Para reactivar la visualización después de haber utilizado el color del fondo basta con especificar con la misma instrucción COLOR el nuevo color que se desea utilizar.
Por ejemplo, en el sistema 2010 Siprel, dotado de pantalla monocromática, sólo puede utilizarse el código 7; por tanto, en este microordenador sólo existen dos posibilidades: COLOR = 0 (no en visión) y COLOR = 7 (en visión).
En la figura de la página siguiente se ha representado el diagrama de flujo de un programa de desplazamiento de un rectángulo. El listado se ha omitido, puesto que es prácticamente idéntico al del programa para el trazado parametrizado de cuadrados y rectángulos.
La rotación presenta algunas complicaciones de naturaleza matemática, debidas a la necesidad de utilizar las fórmulas trigonométricas para obtener los parámetros (coordenadas). En la figura de la página 1425 se ha representado el diagrama de flujo de un programa de ejemplo que activa la rotación de un segmento. El bloque 1000 está constituido por las instrucciones de introducción de los datos iniciales, que son

— la longitud del segmento
— el ángulo inicial con respecto al eje X
— la posición inicial (X0,Y0) del extremo alrededor del cual se tendrá la rotación

El bloque 1500 también puede evitar la visualización del segmento en la posición inicial, saltando directamente al bloque 4000. Al final de la presentación, si el operador desea continuar, el programa vuelve a las instrucciones del bloque

Trazado de segmentos orientados aleatoriamente en una pantalla en colores.



Centro THC/Marka

DIAGRAMA DEL PROGRAMA DE DESPLAZAMIENTO DE UNA FIGURA

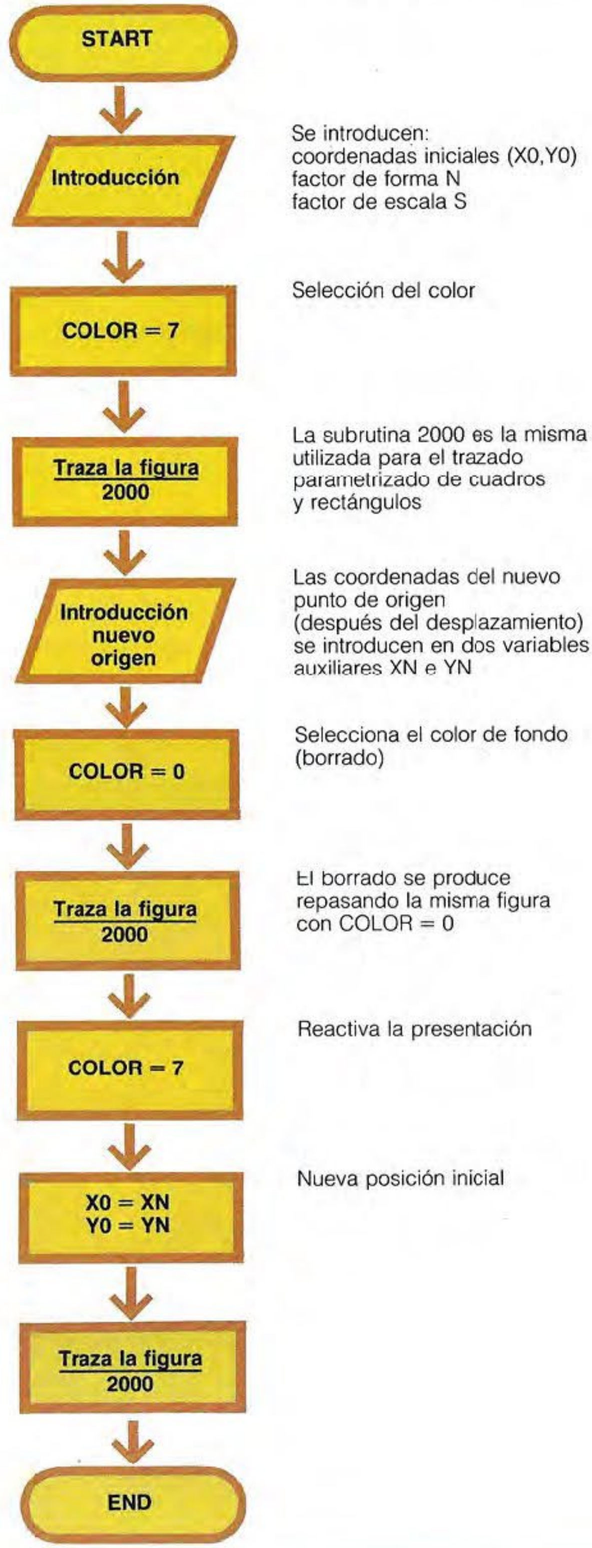
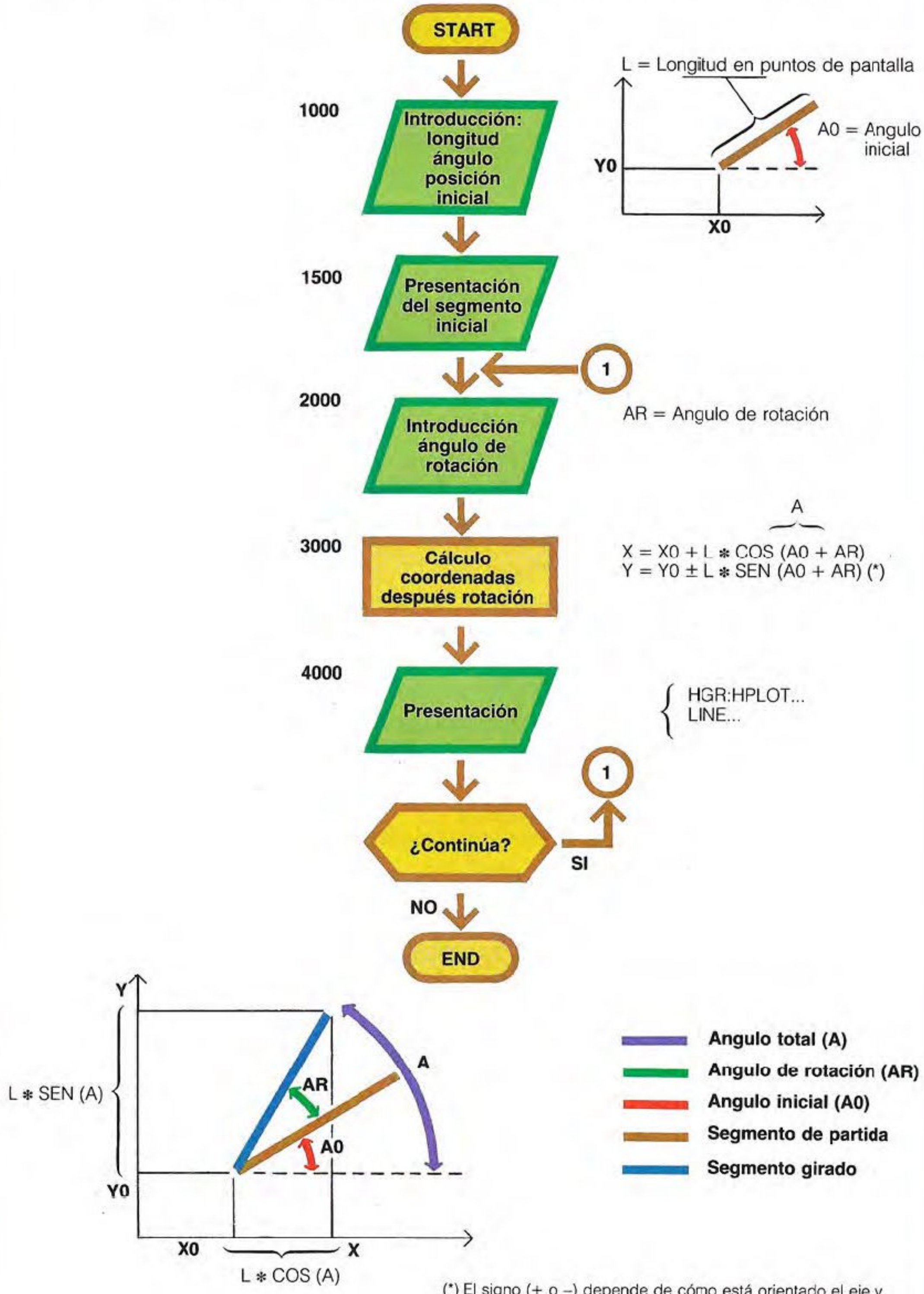


DIAGRAMA DE FLUJO PARA LA ROTACION DE UN SEGMENTO



(*) El signo (+ o –) depende de cómo está orientado el eje y.
Por tanto, es un parámetro característico de la máquina particular.

2000 (punto 1) y se activa el ciclo de introducción del ángulo de rotación, del cálculo con nuevas coordenadas y la visualización. La única diferencia con respecto al caso ya examinado de la traslación consiste en el cálculo de las coordenadas del extremo que gira. Este cálculo, ilustrado siempre en la figura de la página 1425, no presenta dificultades de programación. Las fórmulas de resolución (indicadas al lado del bloque 3000) pueden suponerse válidas salvo la advertencia de utilizar los ángulos expresados en radianes. En el uso corriente, los ángulos se expresan en grados sexagesimales, mientras que en muchos lenguajes de programación, las rutinas de sistema emplean los radianes. La transformación de una unidad de medida a la otra se obtiene con la fórmula

$$\text{Angulo en radianes} = \text{Angulo en grados} * 3.14/180$$

Abajo y en la página siguiente se ha representado el listado del programa en las dos versiones para Siprel 2010 (Apple y compatibles) y Olivetti M20 con la adición de la transformación de grados a radianes. La forma, representada sólo a efectos indicativos, no es la mejor desde el punto de vista de la programación, y puede hacerse más «profesional», efectuando la transformación de grados a radianes y el cálculo de las coordenadas con el uso de las funciones:

```
DEF FNR(AG) = AG * 3.14/180
DEF FNX(L,A) = X0 + L * COS(A)
DEF FNY(L,A) = Y0 + L * SEN(A)
```

(AG = Angulo en grados, A = Angulo en radianes, L = Longitud del segmento).
El ángulo A expresado en radianes se obtiene del ángulo AG expresado en grados, escribiendo simplemente

```
A = FN R(AG)
```

Análogamente, las nuevas coordenadas se ob-

ROTACION DE UN SEGMENTO

Versión Siprel 2010, Apple y compatibles

```
1000 TEXT
1100 HOME
1200 PRINT "Posición inicial";
1300 INPUT X0,Y0
1400 PRINT "Longitud del segmento";
1500 INPUT L
1600 PRINT "Angulo de partida (en grados)";
1700 INPUT A0
1800 HGR: HCOLOR=7
1900 GOSUB 3000
2000 HOME
2100 VTAB22: PRINT "Angulo de rotación (en grados)";
2200 INPUT A1
2300 HCOLOR=0
2400 GOSUB 4000: REM Borra
2500 AR=A1
2600 HCOLOR=7
2700 GOSUB 3000: REM Dibuja
2750 HOME
2800 VTAB 22: PRINT "Quiere continuar (s/n)";
2850 INPUT A$
2900 IF A$="s" THEN GOTO 2000
2950 END
3000 REM Calcula
3100 A0=(A0*3.14)/180
3200 AR=(AR*3.14)/180
3300 X=X0+L*COS(A0+AR)
3400 Y=Y0-L*SEN(A0+AR)
4000 REM Dibuja el segmento
4100 HPLOT X0,Y0 TO X,Y
4200 RETURN
```

Versión Olivetti M20

```
1000 CLEAR
1100 REM No es necesaria
1200 PRINT "Posición inicial";
1300 INPUT X0,Y0
1400 PRINT "Longitud del segmento";
1500 INPUT L
1600 PRINT "Angulo de partida (en grados)";
1700 INPUT A0
1800 CLS: COLOR 1
1900 GOSUB 3000
2000 CURSOR (1,15),1
2100 PRINT "Angulo de rotación (en grados)";
2200 INPUT A1
2210 CURSOR (1,15),1
2220 PRINT "                                        ": REM 40 espacios
2300 COLOR 0
2400 GOSUB 4000: REM Borra
2500 AR=A1
2600 COLOR 1
2700 GOSUB 3000: REM Dibuja
2750 REM No es necesaria
2770 CURSOR (1,15),1
2800 PRINT "Quiere continuar (s/n)";
2850 INPUT A$
2900 IF A$="s" THEN GOTO 2000
2950 END
3000 REM Calcula
3100 A0=(A0*3.14)/180
3200 AR=(AR*3.14)/180
3300 X=X0+L*COS (A0+AR)
3400 Y=Y0+L*SEN(A0+AR)
4000 REM Dibuja el segmento
4100 LINE (X0,Y0) - (X,Y)
4200 RETURN
```

tienen con las instrucciones

X = FN X(L,A)
Y = FN Y(L,A)

Trazado de una circunferencia

En algunas máquinas existen instrucciones que permiten dibujar figuras complejas sin recurrir a subrutinas escritas por el usuario. Un ejemplo ya conocido es la instrucción LINE con la opción B, que en el sistema Olivetti M20 permite obtener el trazado de un rectángulo.
Una forma más compleja es la de la instrucción para dibujar una circunferencia. La sintaxis y las opciones previstas dependen del tipo de máquina y de la versión del Basic utilizados.
En el sistema M20, la instrucción completa es

CIRCLE %N, (X,Y), R, C, A, V

con los significados:

%N número de la ventana (si no se especifica la instrucción, actúa sobre la última seleccionada)
X,Y coordenadas del centro
R radio
C color

El parámetro A es un valor numérico que expresa el factor de forma.
El uso de la opción A está motivado por la necesidad de compensar la diferente densidad de los puntos de la pantalla al pasar del eje X al eje Y, pero en algunos casos puede permitir el trazado de figuras circulares alargadas (similares a elipses). En la máquina indicada, el valor de A que permite obtener una circunferencia es de 0.807; aumentándolo o disminuyéndolo se tiene una deformación según uno u otro de los ejes. Por tanto, el parámetro es opcional, y por omisión se supone el estándar.
El valor V es otro parámetro opcional utilizado con el color, y tiene los mismos significados vistos para la instrucción LINE.
Todos los parámetros citados, excluidos los de las coordenadas del centro y del radio, son opcionales, por lo que la forma más sencilla de la instrucción es

CIRCLE (X,Y),R

Algunas máquinas utilizan otros dos parámetros opcionales que permiten trazar sólo un arco de circunferencia especificando el ángulo de apertura. En estos casos debe proporcionarse el ángulo de inicio y el ángulo de final (expresados en radianes) del ángulo a dibujar. Por ejemplo, la instrucción

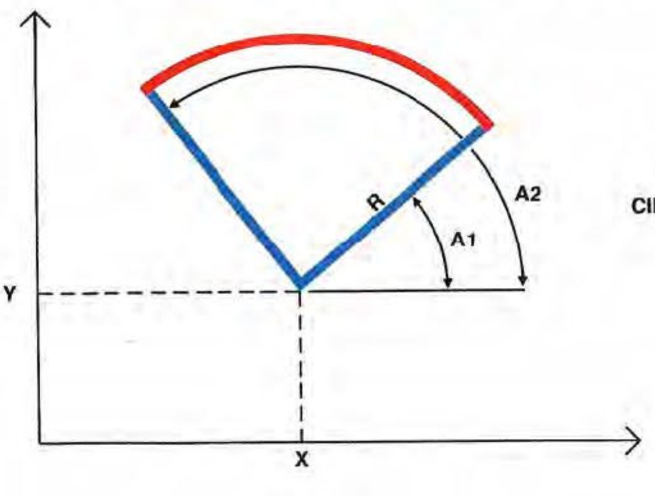
CIRCLE (X,Y),R,A1,A2

dibuja un arco de circunferencia (con centro en X,Y y radio R) que principia por el ángulo A1 y termina una vez alcanzado el ángulo A2. Abajo se ha representado un ejemplo.
En muchas máquinas, la instrucción CIRCLE no está, y en su lugar, el sistema debe utilizar una rutina escrita por el usuario cada vez que debe visualizar una circunferencia. Despreciando el factor de forma, para trazar un círculo es necesario conocer tres parámetros: las coordenadas del centro (X0,Y0) y el radio (R). La parametrización se obtiene calculando las coordenadas de cada punto de la circunferencia en función de los valores de X0, Y0 y R.
En la figura de arriba de la página siguiente se ha representado el método de cálculo para un punto general P de la circunferencia, cuyas coordenadas se obtienen de las fórmulas

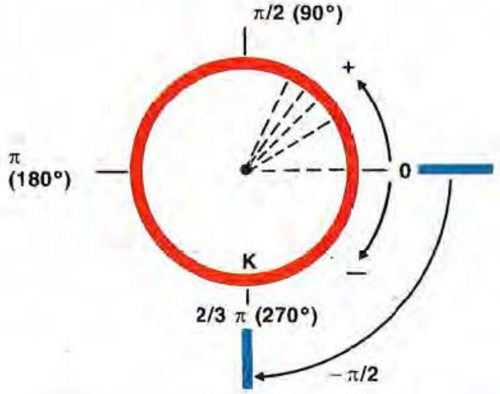
X = X0 + R * COS(A)
Y = Y0 + R * SEN(A)

Fijados los valores de las coordenadas del centro (X0,Y0) y del radio, variando el ángulo A entre 0 y 2π (360 grados sexagesimales), de las fórmulas anteriores se obtienen las coordenadas de todos los puntos de la circunferencia.
Supongamos que se empieza por el punto P1.

INSTRUCCION CIRCLE CON LA ESPECIFICACION DE LOS ANGULOS

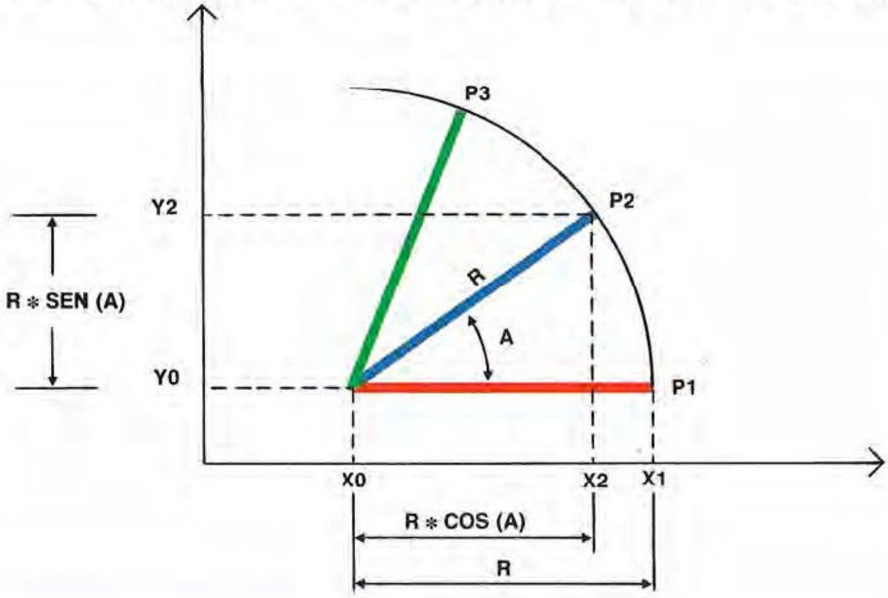


CIRCLE (X,Y), R, A1, A2

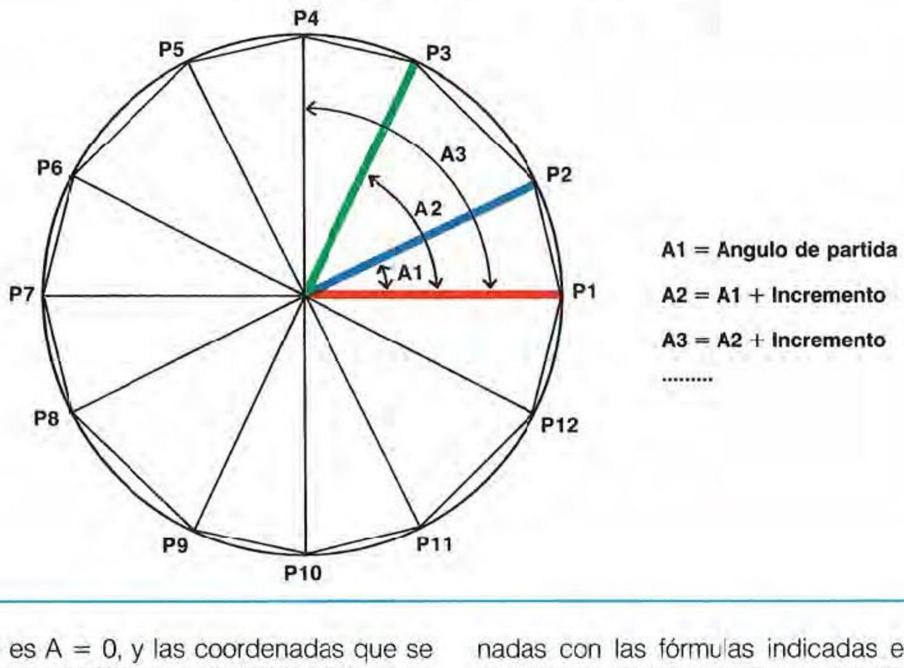


Los ángulos (A1 y A2) se expresan en radianes y referidos a la circunferencia goniométrica. El ángulo 0 está en el extremo derecho y los valores son crecientes en sentido antihorario.
En algunas máquinas pueden utilizarse ángulos negativos.
Por ejemplo, el punto K puede indicarse como + 2/3 π (rotación antihoraria, sentido positivo) o como π/2 (rotación horaria, sentido negativo)

CALCULO COORDENADAS DE UN PUNTO EN UNA CIRCUNFERENCIA



TRAZADO DE UNA CIRCUNFERENCIA



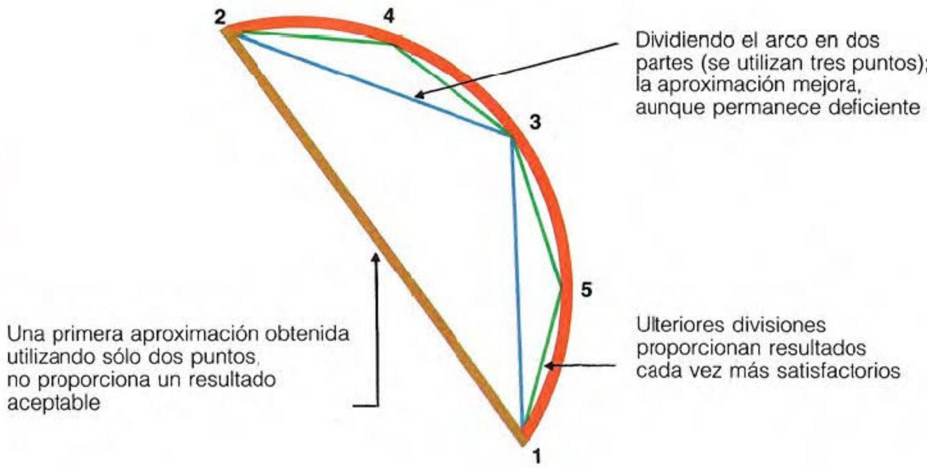
El ángulo es A = 0, y las coordenadas que se obtienen son [COS(0) = 1, SEN(0) = 0]:

X = X0 + R
Y = Y0

A continuación puede incrementarse una cantidad fija el ángulo A, calcular las nuevas coordenadas con las fórmulas indicadas e ir procediendo así. Al final (ver figura de arriba) se obtendrá un conjunto de puntos que pertenecen a la circunferencia de radio R y con centro X,Y; uniéndolos dos a dos con segmentos puede obtenerse una aproximación de la circunferencia. La excesiva distancia existente entre los puntos

trazados en la figura amplifica notablemente la escasa semejanza entre la figura obtenida y una circunferencia. En realidad, el incremento a dar al ángulo A debe ser mucho más pequeño que el indicado; de esta manera, la distancia entre dos puntos calculados consecutivamente disminuye, y el segmento de recta se aproxima mejor al arco de círculo correspondiente (ver gráfico de abajo). Abajo y en la página siguiente se ha representado el listado de un programa para el trazado de una circunferencia en las dos versiones para el Sprel 2010 (Apple y compatibles) y Olivetti M20, también si para la segunda máquina existe una instrucción dedicada. Como puede verse en los listados, las instrucciones a sustituir para pasar de una a otra versión sólo son dos (HGR, que desaparece y HPLOT, que se convierte en LINE...). En el diagrama de flujo de la página 1432 se ha representado una variante que permite trazar un círculo lleno. La modificación consiste solamente en eliminar la igualdad X1 = X2 e Y1 = Y2.

APROXIMACION DE UNA CIRCUNFERENCIA



TRAZADO DE UNA CIRCUNFERENCIA

Versión Siprel 2010, Apple y compatibles

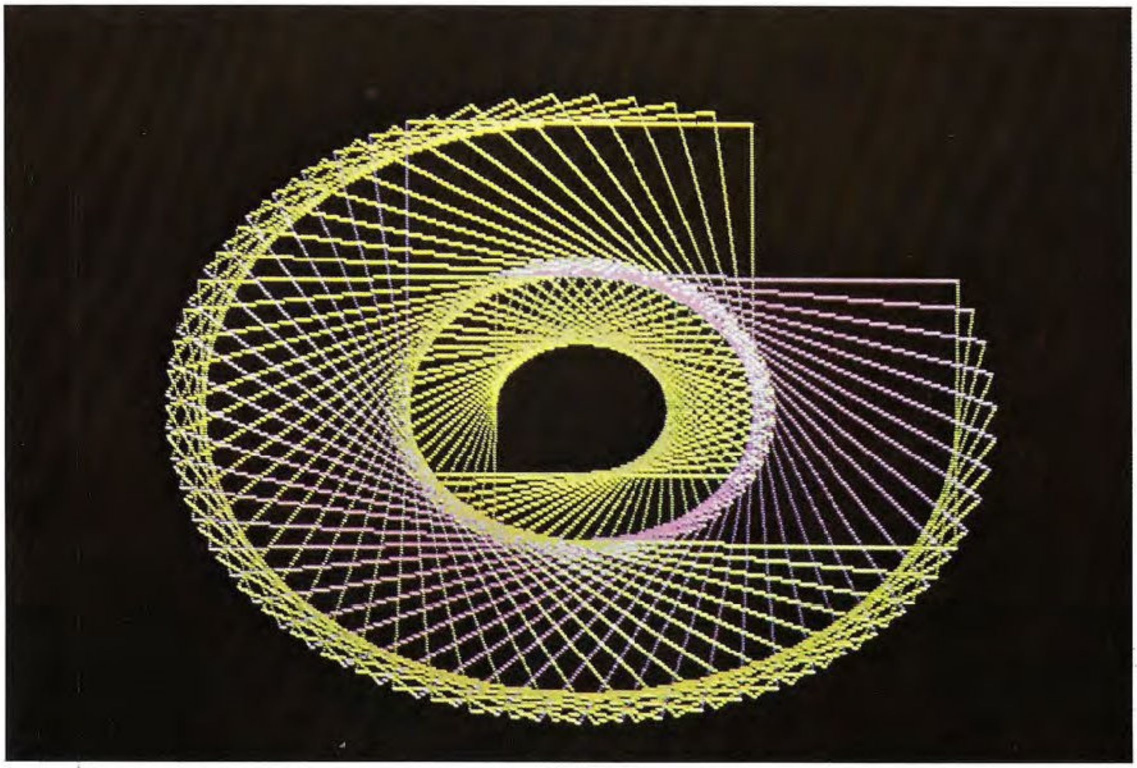
```
10 REM Círculo
20 TEXT
30 HOME
40 PRINT "Coordenadas del centro (X0,Y0)";
50 INPUT X0,Y0
60 PRINT "Longitud del radio";
70 INPUT R
80 PRINT "Paso del incremento";
90 INPUT D
100 HGR HCOLOR=7
110 GOSUB 1000
120 VTAB 22
130 PRINT "Continúa (s/n)";
140 INPUT C$
150 IF C$="s" THEN GOTO 10
160 END
1000 REM Dibuja
1010 X1=X0+R
1020 Y1=Y0
1030 FOR A=0 TO 6.28 STEP D
1040 X2=X0+R*COS(A)
```

```
1050 Y2=YO+R*SEN (A)
1060 HPLOT X1,Y1 TO X2,Y2
1070 X1=X2: Y1=Y2
1080 NEXT A
1090 RETURN
```

Versión Olivetti M20

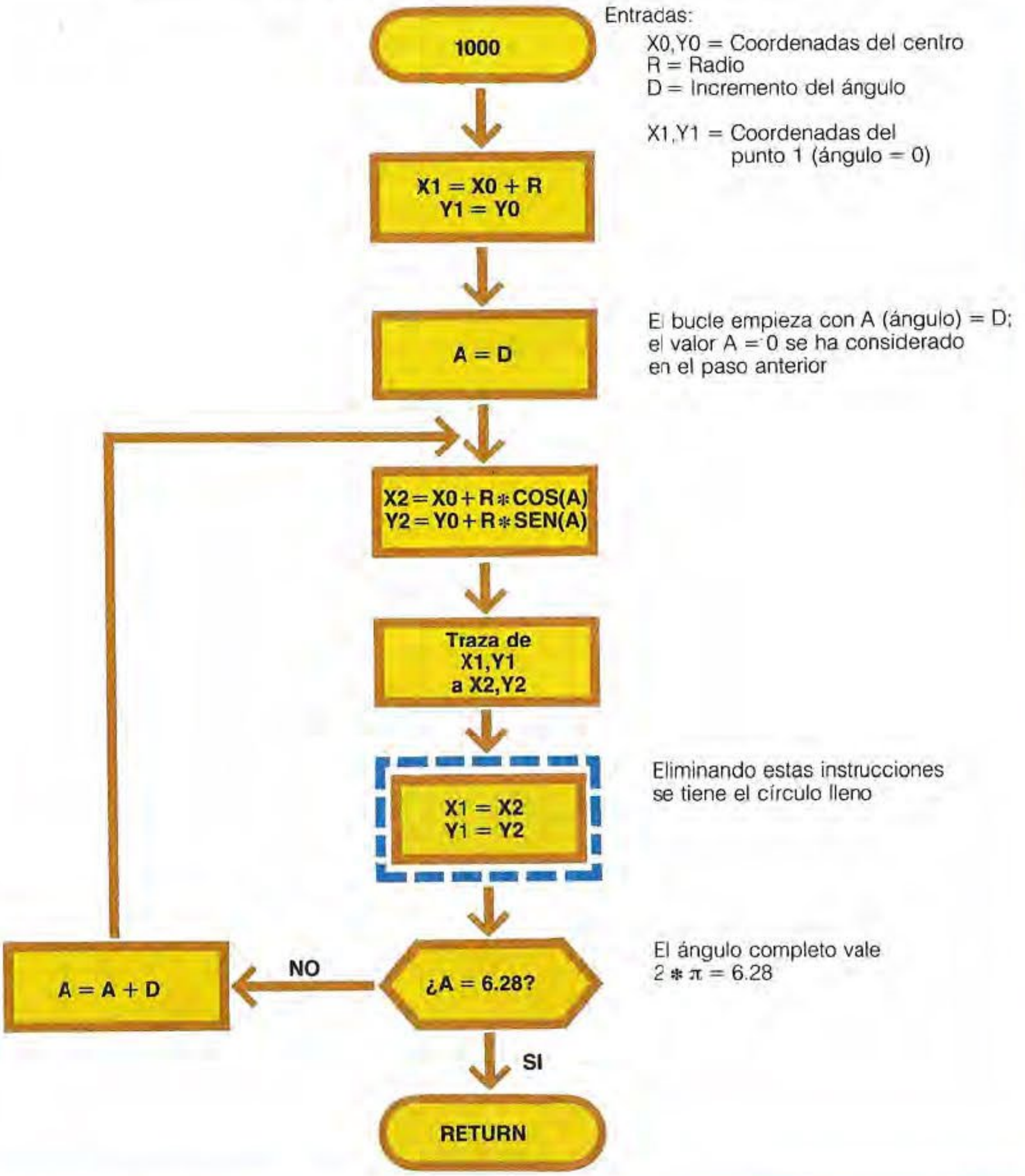
```
10 REM     Círculo
20 REM No es necesaria
30 CLS: COLOR 1
40 PRINT "Coordenadas del centro (XO,YO)";
50 INPUT XO,YO
60 PRINT "Longitud del radio";
70 INPUT R
80 PRINT "Paso del incremento";
90 INPUT D
100 CLS: COLOR 3
110 GOSUB 1000
120 CURSOR (1,15),1
130 PRINT "Continúa (s/n)";
140 INPUT C$
150 IF C$="s" THEN GOTO 10
160 COLOR 1: END
1000 REM Dibuja
1010 X1=XO+R
1020 Y1=YO
1030 FOR A=0 TO 6.28 STEP D
1040 X2=XO+R*COS(A)
1050 Y2=YO+R*SEN(A)
1060 LINE (X1,Y1) - (X2,Y2)
1070 X1=X2: Y1=Y2
1080 NEXT A
1090 RETURN
```

Esta figura se ha obtenido haciendo girar a pasos un rectángulo.



Centro THC/Marka

DIAGRAMA DE FLUJO DE LA SUBRUTINA CIRCULO



Presentación del gráfico de una función

Una función matemática (monodroma) es una ley analítica que hace corresponder a cada valor de la variable independiente X un valor de la variable dependiente Y. Cuando las variables independientes son más de una, la definición es la misma, pero se tendrán más variables (X_1, X_2...). En cambio, cuando el valor asumido por la variable independiente en correspondencia a una cierta configuración de valores de las variables independientes no es uno sólo, se habla de una función de más valores, o polidroma.

El modo usual de representar una función es

$$Y = F(X)$$

con la que se indica que Y es función de la variable independiente X.

Dada una función monodroma (es decir, de un solo valor) de una (sola) variable, el modo más

sencillo para conocer los valores de la variable dependiente consiste en construir una tabla en la que se representan los valores de la X (variable independiente) y los correspondientes valores de la Y, calculados con las operaciones matemáticas expresadas por la función.

En muchos casos, la tabla de los valores no proporciona una visión inmediata de la relación X/Y; es mucho más adecuada la representación gráfica que se obtiene indicando en un papel los valores de la tabla, normalmente los valores de X en la horizontal y los de Y en vertical. El dibujo que resulta es el gráfico de la función.

Trazado del gráfico de una función por segmentos

El ordenador puede realizar el trazado del gráfico de una función utilizando la única instrucción gráfica del trazado de un segmento.

El método, ilustrado en la página siguiente, consiste en generar un conjunto de valores de la variable independiente, calcular después cada valor de la variable dependiente y unir los puntos así hallados.

El primer bloque se dedica a la introducción por teclado de los valores inicial (XI) y final (XF) de la variable independiente y del paso de variación P. El primer punto tendrá las coordenadas X = XI e Y = f(XI), el siguiente X = XI + P e Y = f(XI + P), y así seguidamente incrementado siempre X con la cantidad P y calculando el correspondiente valor de Y. El dibujo del gráfico viene dado por la línea de trazos que une los puntos así determinados.

La figura que resulta es de trazos, o sea una aproximación de la función Y = F(X), obtenida mediante segmentos. La representación será tanto más aproximada a la verdadera forma de la función como cuanto más cercanos estén los puntos, o sea como cuanto más pequeño sea el paso P. El valor óptimo de P no puede establecerse a priori, sino que depende del tipo de curva que se está representando, o sea de la forma particular de la función Y = F(X). Por ejemplo, si la función es la que describe una recta (supongamos Y = 3 * X + 5) son suficientes sólo dos puntos, puesto que por dos puntos sólo pasa una recta. Uniendo las coordenadas de los dos extremos de la recta con un segmento, se tiene la seguridad de haber representado exactamente la función. En cambio, si la función tiene una forma compleja y es necesario subdividirla en pequeñas partes, cada una de las cuales puede aproximarse bien mediante un segmento, con una lógica muy similar a la ya utilizada para el trazado de una circunferencia [las coordenadas X,Y de cada punto de una circunferencia están ligadas por una relación del tipo $X^2 + Y^2 = R^2$, que puede reducirse a $Y = \sqrt{R^2 - X^2}$, que es una forma de Y = F(X)]. Anteriormente, para determinar los valores de X e Y, se han adoptado las fórmulas trigonométricas, pero los dos aspectos son del todo idénticos y se deducen de dos métodos diferentes de solución de un triángulo rectángulo (por el teorema de Pitágoras o con las fórmulas trigonométricas).

En el diagrama de flujo de la página siguiente se calculan las coordenadas del primer punto (X1,Y1) y las del siguiente (X2,Y2), obtenido con un incremento P sobre el eje X (X2 = X1 + P). A continuación, el punto de llegada anterior (X2,Y2) se convierte en el de partida (por lo que deben modificarse los valores X1 e Y1 en el modo X1 = X2 e Y1 = Y2); la nueva posición final (nuevos valores de X2,Y2) se obtiene siempre incrementando X1 con la cantidad P y calculando, con la expresión algebraica que representa la función, el correspondiente valor de Y (Y2). El procedimiento debe repetirse para todo el intervalo del gráfico, o sea hasta que X2 alcanza XF (XF es el valor final introducido por el usuario).

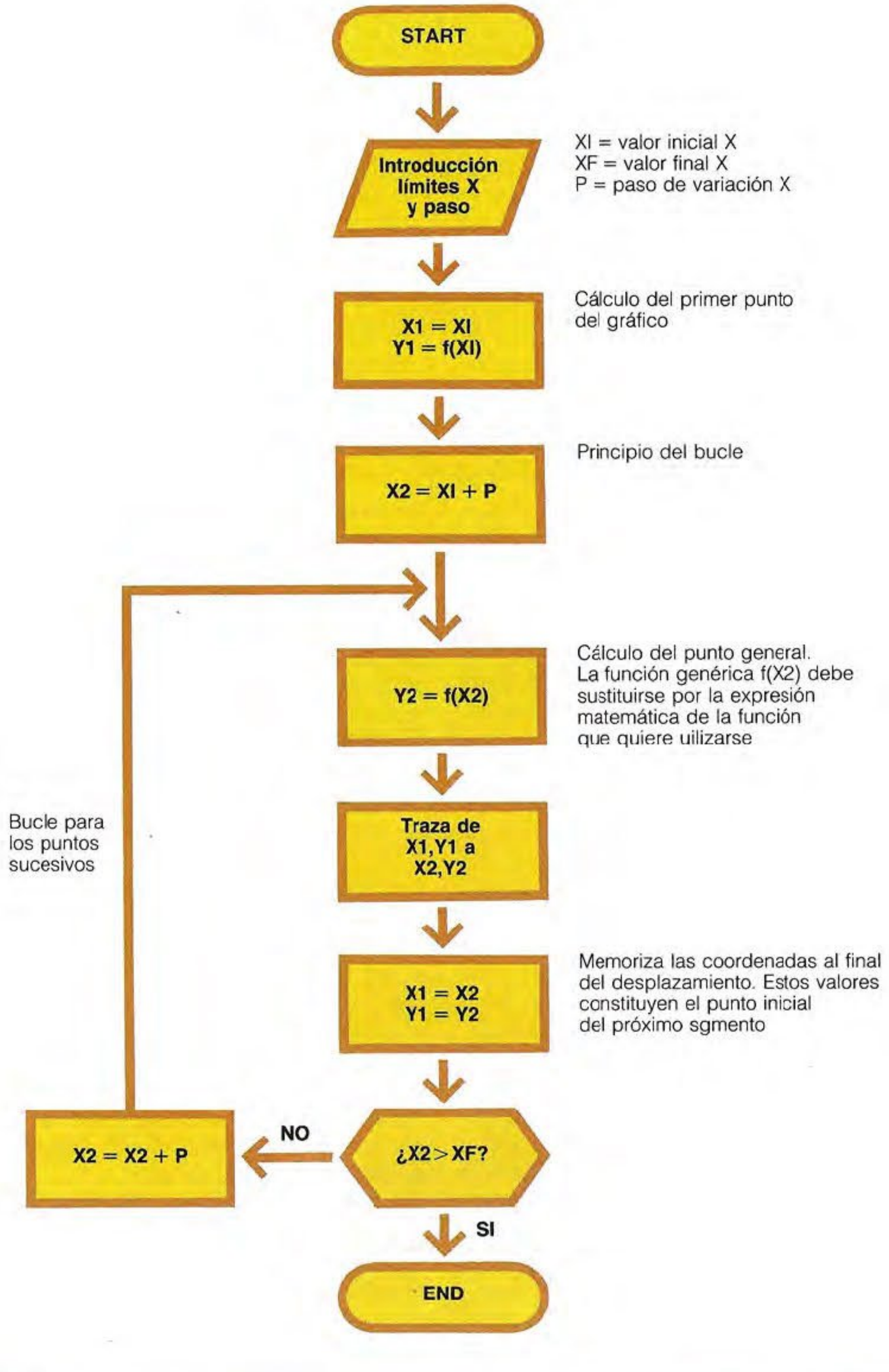
El procedimiento puede activarse con un bucle sobre la variable X en el intervalo XI, XF con paso P. Normalmente, para simplificar el listado, el primer punto [X1 = XI, Y1 = f(XI)] se calcula fuera del bucle, que tiene como primer extremo el valor XI + P.

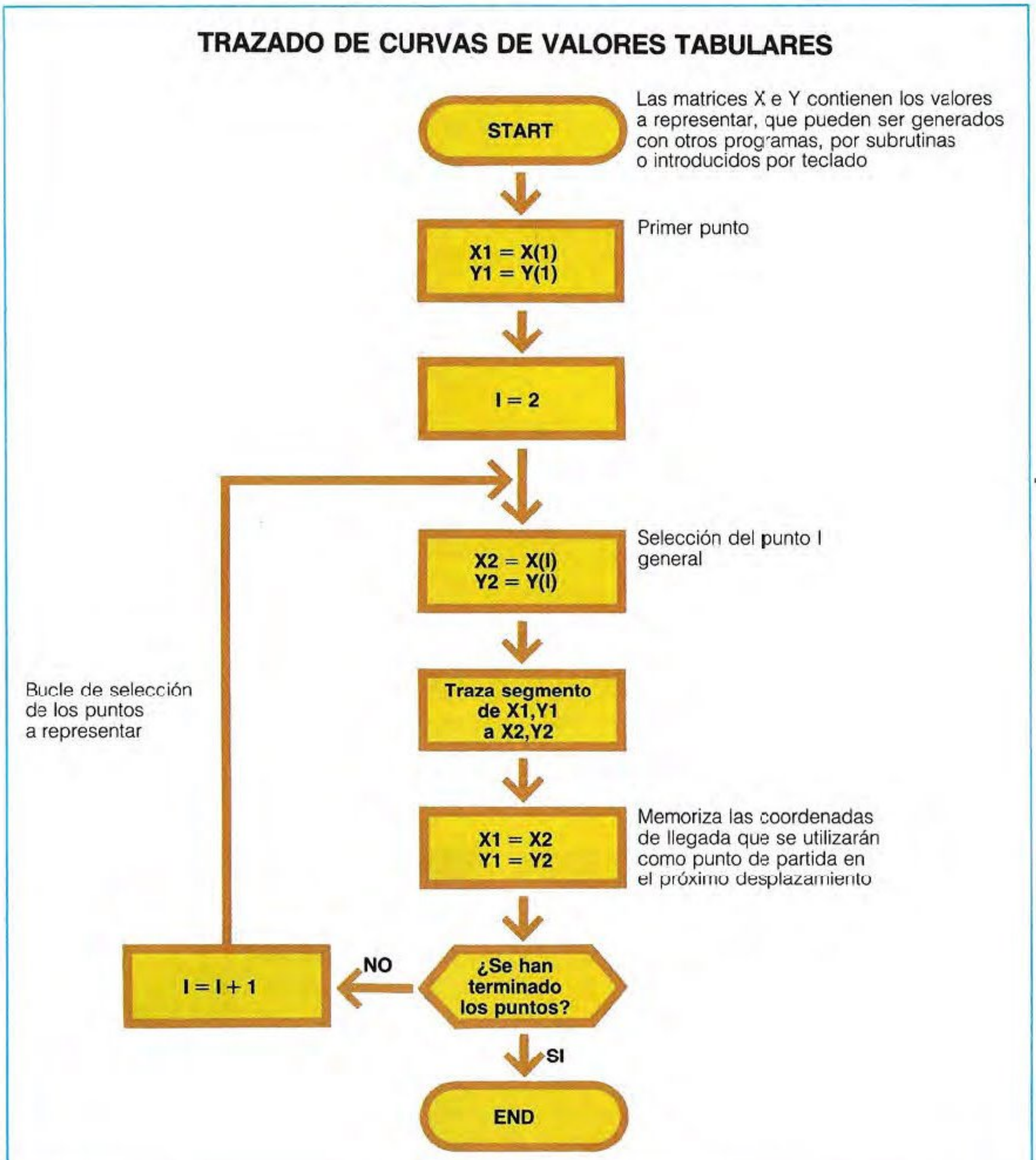
El método ilustrado sólo es válido en líneas generales. Para obtener una buena rutina de trazado deben tenerser en cuenta diversos factores. El primero es la tipología de la función. A priori no se ha dicho que una función pueda ser representada con una ley matemática. Éste es el caso en que debe representarse una tabla. Los valores X e Y se transferirán a dos variables dimensionadas y el programa sólo deberá tomarlos y utilizarlos. En la página 1435 se ha representado el diagrama de flujo a adoptar para este tipo de problema gráfico.

No existe ninguna diferencia sustancial con respecto al método anterior.

El bucle no se realiza incrementando la X, puesto que en los valores de la tabla no se tiene necesariamente una variación regular (por ejemplo podrían existir los valores 2 y 5 y no existir los valores 3 y 4); en cambio, se incrementa el índi-

GRAFICO DE UNA FUNCION. DIAGRAMA DE PRINCIPIO





ce I que apunta a los elementos de la tabla. De esta manera, poniendo por ejemplo I = 2 se extraen las coordenadas del segundo elemento, con I = 3 las del tercero, etc. La otra diferencia con respecto al diagrama de flujo anterior es la ausencia del cálculo Y2 = f(X2), sustituido por una simple transferencia de variable [X2 = X(I), Y2 = Y(I), habiendo indicado con X(I) e Y(I) los elementos de la tabla de la posición I].

Otros factores a considerar son los órdenes de magnitud y los errores de interpolación. Los valores a representar pueden tener órdenes de magnitud variable. En algunos casos, los puntos pueden estar tan cercanos que no pueden distinguirse y en otros pueden caer fuera de las dimensiones máximas de la pantalla. Para obtener en cada uno de los casos valores representables sobre la pantalla deben representarse no los valores reales, sino valores que sean proporcionales según un factor de escala.

Al trazar un gráfico representable con una función no pueden utilizarse todos los puntos posi-

bles, sino algunos de ellos; entre un punto y el siguiente suele suponerse un proceso rectilíneo. Si dos puntos consecutivos están muy separados, esta interpolación puede conducir a una representación inexacta, y normalmente muy diferente del proceso real de la función. La aproximación resultante es análoga a aquella en que se ha trazado una circunferencia por segmento, con la diferencia de que en el caso de la circunferencia se conoce a priori la forma exacta y el gráfico no puede generar errores de interpretación, mientras que en el caso de curvas generales, la forma no se conoce, y el error puede ser importante.

Trazado de una recta

La función más sencilla a representar es la recta. La ecuación general de una recta, o sea la ley matemática que permite el cálculo de los valores de la Y en función de los de la X, es

$$Y = A * X + B$$

con A y B valores constantes.
El gráfico puede obtenerse con el mismo programa ilustrado en la página 1434, pero existe una forma más elegante, consistente en definir una función de usuario igual a la de la recta; el cálculo de los valores Y se producirá entonces simplemente llamando la función con los valores de X. Al principio del programa deberá insertarse la definición de la función, indicada por ejemplo con la letra R:

```
DEF FN R(X) = A * X + B
```

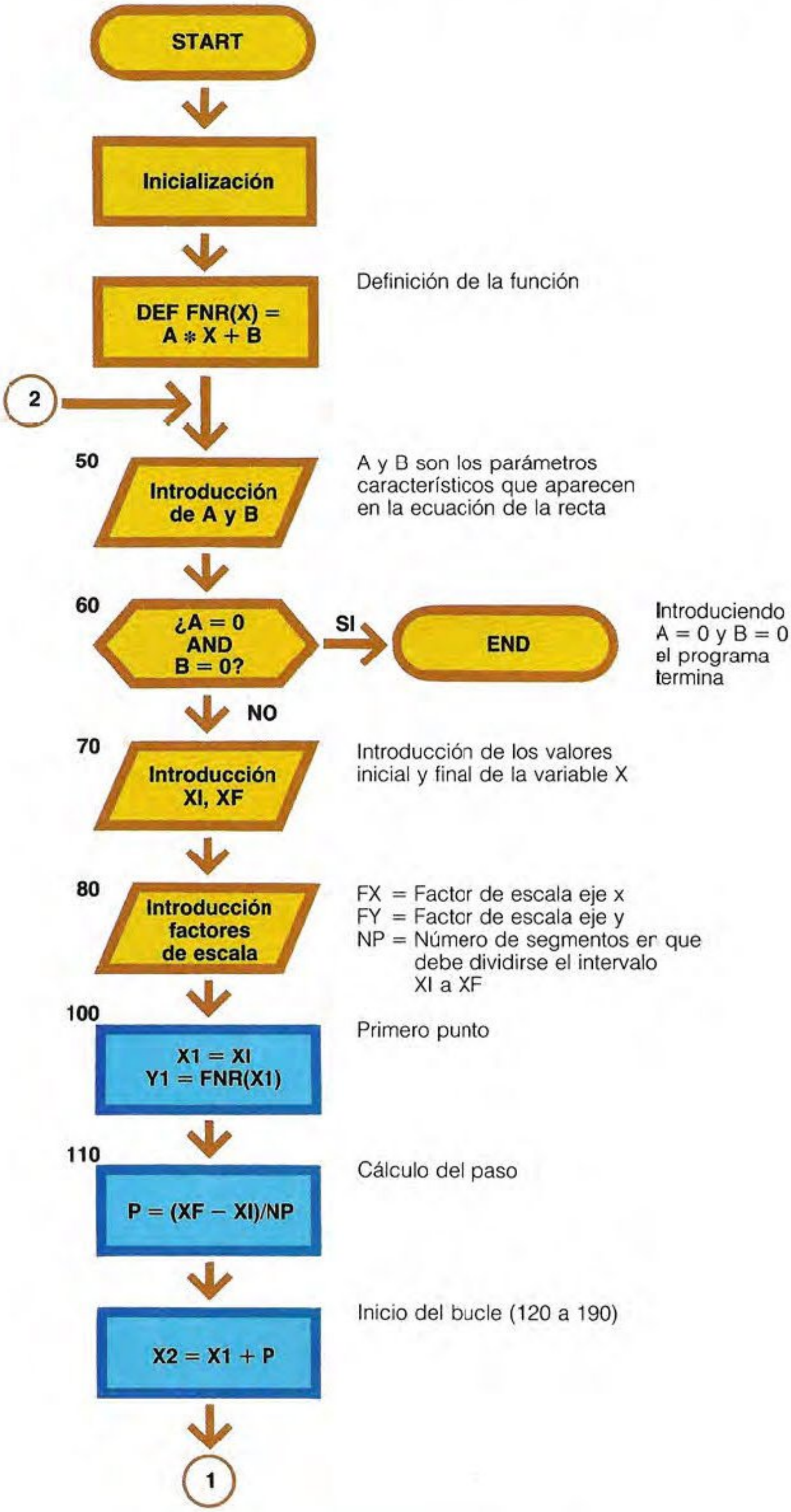
En este caso específico (recta), la sencillez de la fórmula no hace particularmente útil la definición de una función de usuario, pero con funciones más complejas es totalmente aconsejable. Para estas últimas, el método ofrece diversas ventajas, debidas al hecho de que la función está escrita en un solo punto del programa y puede modificarse fácilmente; además, todas las rutinas gráficas se hacen parametrizadas, puesto que se refieren a una función general FN R(X). En las páginas 1437 y 1438 se han indicado el diagrama de flujo de un programa que dibuja una recta (los parámetros A y B se introducen por teclado) para un intervalo dado de valores XI y XF; el correspondiente listado se ha representado en las páginas 1438 y 1439. El programa puede transportarse a otras máquinas variando la sola instrucción HPLOT.

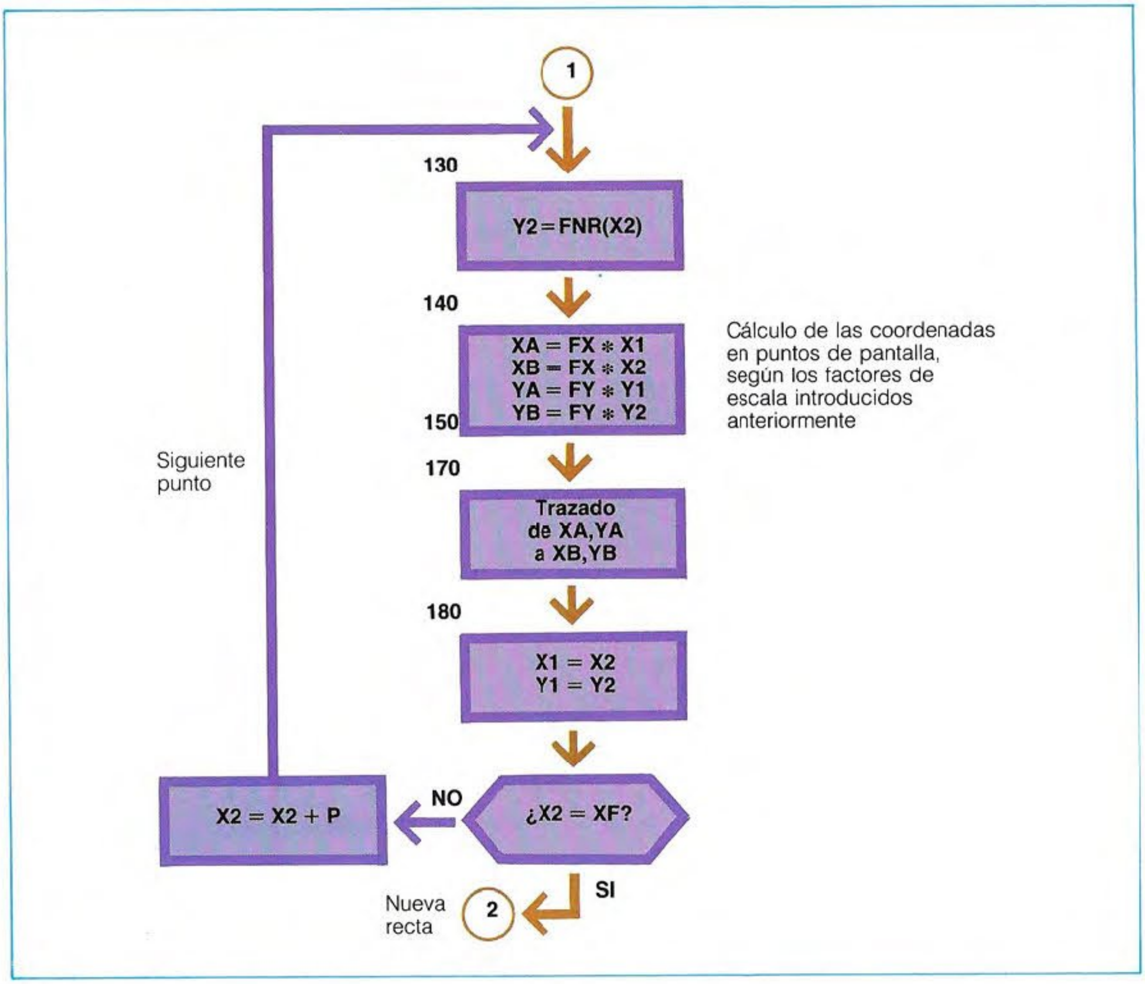
Aplicación profesional del ordenador para gráficos.



J. Zainer/Marka-Int'l Stock Photo

DIAGRAMA DE FLUJO PARA EL TRAZADO DE UNA RECTA





TRAZADO DE UNA RECTA

Versión Siprel 2010, Apple y compatibles

```
10      HOME
20      N = 5
      : S = 153
      : E = 265
      : O = 15
30    : XO = 140
      : YO = 80
40      DEF FN R(X) = A * X + B
50      VTAB 23
      : INPUT "A,B: ";A,B
60      IF A = 0 AND B = 0 THEN END
70      INPUT "XI,XF: ";XI,XF
80      INPUT "FX,FY,NP: ";FX,FY,NP
90      GOSUB 1000
100     X1 = XI
      : Y1 = FN R(X1)
110   : P = (XF - XI) / NP
120     FOR X2 = X1 + P TO XF STEP P
130     Y2 = FN R(X2)
140     XA = XO + FX * X1
      : YA = YO - FY * Y1
150     XB = XO + FX * X2
      : YB = YO - FY * Y2
160   : IF XA > 265 OR XA < 15 OR XB > 26
           5 OR XB < 15 OR YA > 153 OR YA
```

```
        < 5 OR YB > 153 OR YB < 5
          THEN 180
170     HPLOT XA,YA TO XB,YB
180     X1 = X2
      : Y1 = Y2
190     NEXT
200     HOME
210     VTAB 23
220     INPUT "¿CONTINUO? (S/N) ";S$
230     IF S$ = "S" THEN RUN
240     END
997     REM----------------
998     REM DIBUJA EJES
999     REM----------------
1000    HGR
      : HCOLOR= 3
1010    HPLOT O - 15,N - 5 TO E + 14,N
          - 5 TO E + 14,S + 3 TO O - 15,S
          + 3 TO O - 15,N - 5
1020    HPLOT O,Y0 TO E,Y0
      : HPLOT X0,S TO X0,N
1030    N2 = N
      : FOR N1 = 2 TO 0 STEP - 1
      : HPLOT X0 - N1,N2 TO X0 + N1,N2
      : N2 = N2 - 1
      : NEXT N1
1040    N2 = E
      : FOR N1 = 2 TO 0 STEP - 1
      : HPLOT N2,Y0 - N1 TO N2,Y0 + N1
      : N2 = N2 + 1
      : NEXT N1
1050    HPLOT E + 6,Y0 - 2 TO E + 11,Y0
          + 3
      : HPLOT E + 11,Y0 - 2 TO E + 6,Y0
          + 3
1060    HPLOT X0 - 10,N - 2 TO X0 - 7,N
          + 1
      : HPLOT X0 - 4,N - 2 TO X0 - 10,N
          + 3
1070    IF FX = 1 THEN 1140
1080    FOR G = X0 TO 0 STEP - FX
1090    HPLOT G,Y0 + 1 TO G,Y0 - 1
1100    NEXT G
1110    FOR G = X0 TO E STEP FX
1120    HPLOT G,Y0 + 1 TO G,Y0 - 1
1130    NEXT G
1140    IF FY = 1 THEN 1210
1150    FOR G = Y0 TO N STEP - FY
1160    HPLOT X0 + 1,G TO X0 - 1,G
1170    NEXT G
1180    FOR G = Y0 TO S STEP FY
1190    HPLOT X0 + 1,G TO X0 - 1,G
1200    NEXT G
1210    RETURN
```

La diferencia principal con respecto a los diagramas de flujo anteriores consiste en la presencia de los dos parámetros de escala FX y FY y del cálculo del paso P.

Los factores de escala (FX,FY) son necesarios para llevar los valores de la X y de la Y a cantidades contenidas en las dimensiones de la pantalla. Por ejemplo, si el intervalo en el eje X es 500 (p.e., XI = 0, XF = 500), con una pantalla de 265 puntos horizonales debe utilizarse un factor de escala 0.5. De esta manera, el valor 500 se representaría como si fuese 500 * 0.5 = 250, y queda contenido en el campo de los 265 puntos disponibles.

En general, el factor de escala según el eje X puede calcularse con la expresión

$$FX = \frac{XF - XI}{\text{Número de puntos de pantalla disponibles}}$$

(según el eje Y vale la expresión análoga).

Debe tenerse en cuenta que el valor del cociente es, la mayoría de las veces, un número con algunos decimales y, por tanto, el factor de es-

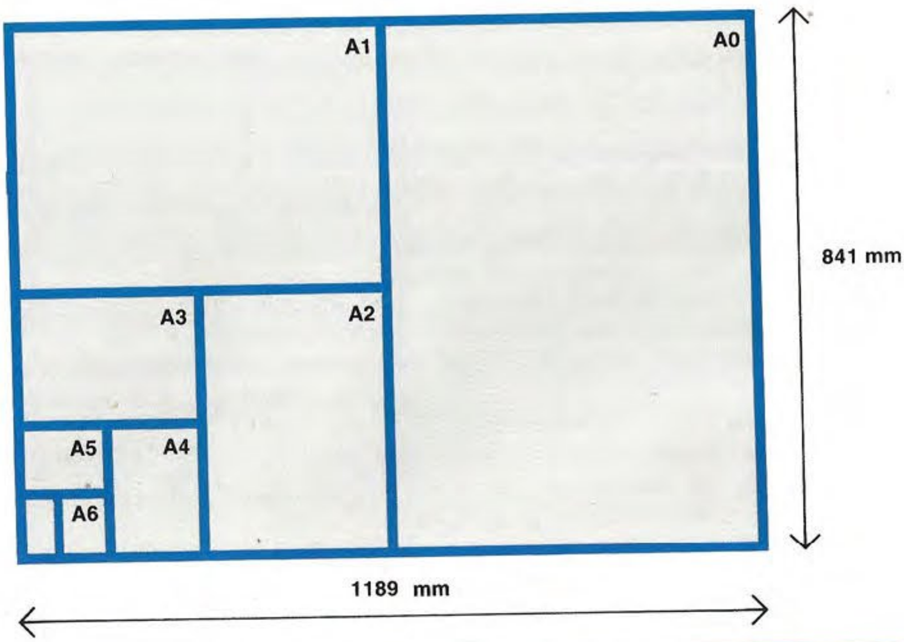
cala que se deduce no permite una lectura rápida del gráfico. En estos casos conviene utilizar la fracción entera más próxima: a cambio de una reducción de las proporciones del gráfico utilizado se tendrá una interpretación más sencilla. En el ejemplo anterior, el factor de escala calculado con la expresión indicada sería 265/500 = 0.53, y para poder interpretar el gráfico, cada lectura debería multiplicarse por 1.886 (1/0.53). Aproximando este valor a 2, se tiene una reducción del campo útil (los puntos utilizados no son 265, sino 250), con la ventaja de poder realizar los cálculos de escala mentalmente, o sea una lectura inmediata del gráfico.

La elección de una escala de representación no es arbitraria como puede parecer. Por el contrario, existen normas bien precisas que regulan las aplicaciones en el sector técnico y que permiten formas de estandarización indispensables en algunas aplicaciones. Por ejemplo, utilizando un plotter o una mesa gráfica sucede que el plano de trabajo tiene dimensiones suficientes para aceptar hojas de papel de formato comercial. Los formatos estándar definidos por las normas DIN se identifican por una sigla que comprende la letra A seguida de un número (por ejemplo A0, A1, A2, etc.). El formato más utilizado es el A4. Los diferentes formatos se obtienen dividiendo sucesivamente por dos el formato de base A0, que por definición tiene una superficie de 1 m^2. En la figura de abajo se han indicado las medidas.

En el listado de las páginas 1438 y 1439 se ha insertado la rutina 1000 que sirve para el trazado de los ejes cartesianos, y que presentaremos más adelante.

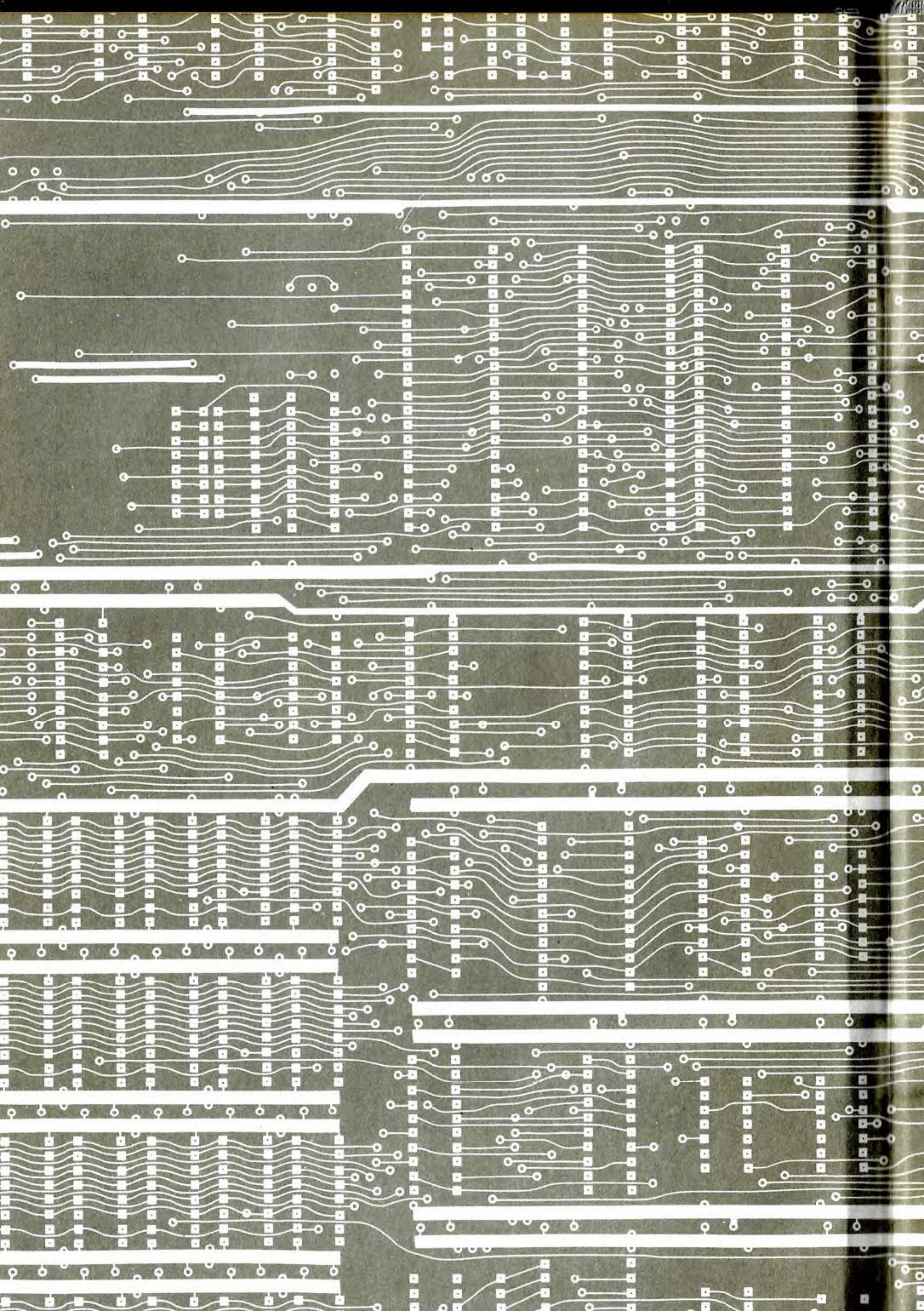
FORMATOS ESTANDAR SEGUN LAS NORMAS DIN

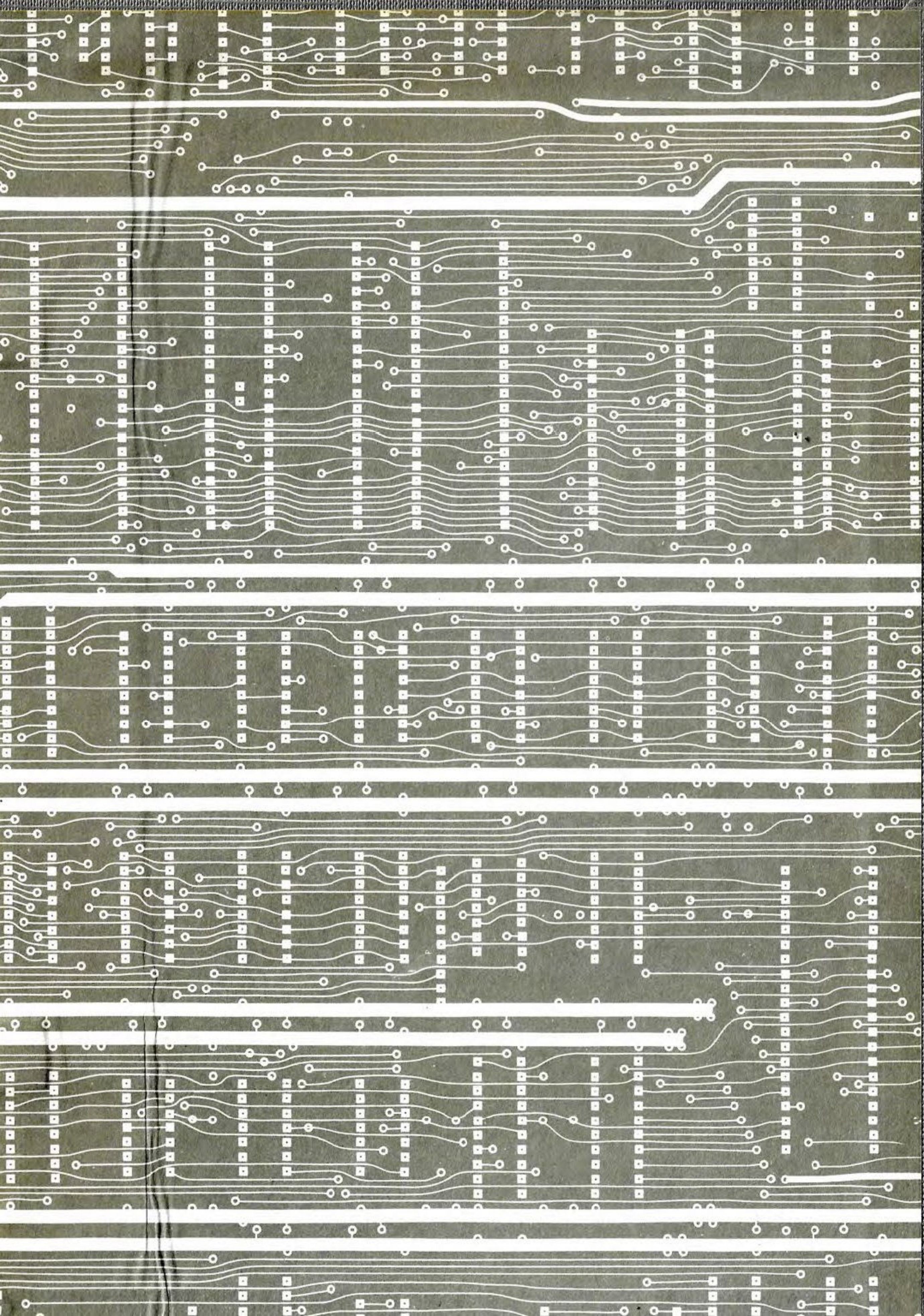
Sigla del formato	Dimensiones en mm
A0	841 × 1189
A1	594 × 841
A2	420 × 594
A3	297 × 420
A4	210 × 297
A5	148 × 210
A6	105 × 148



s
o
-
a
.
s
e
,
el
-
e
e
o

a
a-
s





EDICIONES
FORUM

5

BASIC

ENCICLOPEDIA DE LA INFORMATICA
MINIORDENADORES Y ORDENADORES PERSONALES

